Titre: Title:	Outil de vérification in-situ pour le prototypage de systèmes électroniques
Auteur: Author:	Sylvain Charasse
Date:	2013
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Charasse, S. (2013). Outil de vérification in-situ pour le prototypage de systèmes électroniques [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/1332/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/1332/
Directeurs de recherche: Advisors:	Yvon Savaria, & Yves Blaquière
Programme: Program:	génie électrique

UNIVERSITÉ DE MONTRÉAL

OUTIL DE VÉRIFICATION IN-SITU POUR LE PROTOTYPAGE DE SYSTÈMES ÉLECTRONIQUES

SYLVAIN CHARASSE

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

DÉCEMBRE 2013

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

OUTIL DE VÉRIFICATION IN-SITU POUR LE PROTOTYPAGE DE SYSTÈMES ÉLECTRONIQUES

présenté par : CHARASSE Sylvain
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. AUDET Yves, Ph.D., président
M. SAVARIA Yvon, Ph.D., membre et directeur de recherche
M. BLAQUIÈRE Yves, Ph.D., membre et codirecteur de recherche
M. BOIS Guy, Ph.D., membre

DÉDICACE

À Rémi CHARASSE

REMERCIEMENTS

Je souhaite tout d'abord remercier Yvon Savaria et Yves Blaquière qui m'ont accepté sous leur direction pour mener ce projet de maîtrise. Je leur suis reconnaissant pour m'avoir donné des responsabilités dans mes travaux qui m'ont fait acquérir une meilleure autonomie et des compétences qui se maintiendront tout au long de ma vie professionnelle. Je leur suis aussi reconnaissant pour la richesse de l'expérience et de l'expertise scientifique qu'ils ont su me transmettre durant mes travaux de recherche.

J'aimerais ensuite remercier Pierre Popovic qui est un collègue des plus appréciable et qui, proche de l'ensemble du personnel du génie électrique, a su m'aider dans mes démarches administratives et dans mes travaux.

Je tiens à remercier Guy Bois et Yves Audet pour avoir accepté de me prêter leur temps précieux pour faire partie du jury qui m'évaluera et me donnera un dernier retour sur mon travail.

Enfin, je tiens à remercier toutes les personnes qui ont été proches de moi dans ma vie professionnelle et personnelle, m'accompagnant dans les bons et les mauvais moments, plus précisément :

Le corps administratif du GRM, toujours à l'écoute et prêt à aider les étudiants qui ne comprennent pas tout au système de l'école.

Le corps technique du GRM, donnant des conseils ou des services aux étudiants dans le besoin.

Mes camarades d'école : Michel Gémieux, Gontran Sion, Thalie Keklikian, Keven Chaussé, Lloyd Salvant, Safa Berrima, David Nguyen.

Mes amis d'ici et mes amis d'ailleurs. *(Mention spéciale à « Laloutre » et « Kikill »)*

Le 4436.

Ma famille qui même à distance m'a donné tant d'énergie.

RÉSUMÉ

Les travaux de ce mémoire s'inscrivent dans le cadre d'un projet de recherche appelé DreamWafer™. Le résultat visé est un produit à commercialiser sous le nom WaferBoard™, qui est une plateforme de prototypage rapide de systèmes électroniques visant à réduire leur temps de conception. Au cœur de cette plateforme se retrouve une surface dense de contacts reliés par un réseau d'interconnexions configurables. Pour créer un prototype, le concepteur n'a qu'à y déposer les puces de son système électronique et ces dernières seront dynamiquement interconnectées.

D'abord, les tests et la validation d'un module du WaferBoard™ sont présentés. Il s'agit d'un assemblage de deux prototypes, appelé MiniWaferIC™, constitué de la surface de contacts et de son circuit d'alimentation et de configuration. Les travaux effectués ont permis d'identifier les caractéristiques électriques, des erreurs de conception ainsi que des problèmes d'assemblage et de fabrication. Un banc de test spécifique au MiniWaferIC™ a été mis en place, il a notamment permis de valider une fonctionnalité essentielle: la construction dynamique de chemins de configuration dans le prototype. Cette fonctionnalité permet de configurer des interconnexions entre les contacts disponibles à la surface du MiniWaferIC™.

Ensuite, pour que le prototype d'un concepteur puisse être validé sur le WaferBoard™, il est nécessaire d'avoir accès à un outil pour procéder à la vérification de systèmes électroniques. Une revue de littérature est proposée sur le sujet des méthodes de vérification pour les systèmes électroniques. Elle se concentre sur la vérification des circuits numériques et présente les limites et avantages de l'utilisation de solutions de prototypage.

Enfin, il sera démontré qu'un outil de vérification, in-situ et reconfigurable, peut être directement placé sur le WaferBoard™. Étant ainsi le plus près possible des circuits intégrés ciblés, il est capable de fonctionner à plus haute fréquence et d'effectuer de meilleures mesures de délais. De plus, il est possible d'utiliser et de coupler plusieurs outils de vérification dans un large système pour augmenter la bande passante et l'observabilité. La preuve de concept présentée est une mise en œuvre sur circuit configurable (FPGA) qui est capable de contrôler ou d'observer jusqu'à 64 signaux à une fréquence de 156.25 MHz. Son architecture permet l'accès en parallèle aux signaux, diminuant ainsi le temps de vérification par un facteur de 32 par rapport à un accès sériel.

ABSTRACT

The research work performed as part of this master thesis targets the development of a product, the WaferBoard™, a rapid prototyping platform for electronic systems which allows reducing development time. This platform is based on a wafer scale integrated circuit that implements a configurable interconnect network and that is covered with a dense array of active contacts. To create a prototype, designers just have to put the chips that compose the target system in the prototyping platform that will detect pins locations of user integrated circuits and interconnect them dynamically to implement a user specified netlist.

The first part of this thesis presents tests and verification of a key module of the WaferBoard™. This module is a combination of two modules called the PowerBlock™ and the MiniWaferIC™. It implements an array of active contacts, a configurable interconnect network, programmable power supply devices and related configuration circuits. The experimental work reported in this thesis confirms the functionality of all tested features of the MiniWaferIC™. A custom test bench has been set up to validate the module. Dynamic construction of JTAG configuration paths in the prototype is the main functionality that was validated. Once established, this scan chain is needed to configure interconnections between active contacts of the MiniWaferIC™.

Once a user specified prototype system is implemented with the WaferBoard™, a validation method supported by suitable tool is needed. Before proposing such a method, a review of electronic system verification and validation methods is presented. The review focuses on digital circuit verification and validation methods and techniques. This review allows identifying limits of existing validation methods. A validation method that could take advantage of the WaferBoard™ is then proposed. This validation method and its supporting tools are advantageous as they are based on in-situ configurable logic. In-situ debug and validation circuits, being close to the integrated circuits of the user specified system, can run at higher frequencies and perform better delay measurements. In addition, it is possible to use and to couple multiple verification tools in a large system to increase bandwidth and observability. In a proof of concept that is presented, it is possible to control and observe up to 64 signals at 156.25 MHz. Its architecture featuring a parallel access to the signals decreases the verification time by a factor of 32 compared to a serial access.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ASIC	Application Specific Integrated Circuit
CTL	Core Test Language
DDR2	Double Data Rate 2
DFD	Design for Debug
DFT	Design for Test
DRC	Design Rules Check
EXTEST	Mode de test P1500 : test des interfaces entre plusieurs circuits logiques
FPGA	Field Programmable Gate Array
INTEST	Mode de test P1500 : test de la circuiterie interne d'un circuit logique
ITRS	International Technology Road Map for Semiconductors
JTAG	Join Test Action Group
MCB	Memory Controller Block : bloc de contrôleur de mémoire de Xilinx
P1500	autre nom pour l'IEEE1500
TAM	Test Access Mechanism
ULPI	UTMI+ Low Pin Interface
WBR	Wrapper Boundary Register, enveloppe de cellules de l'enveloppe P1500 (autour d'un circuit logique)
WIR	Wrapper Instruction Register, registre d'instruction de l'enveloppe P1500
WPI	Wrapper Parallel In, port d'entrée parallèle de l'enveloppe P1500
WPO	Wrapper Parallel Out, port de sortie parallèle de l'enveloppe P1500
wrapper	nom très couramment utilisé pour désigner un module IEEE1500 ou l'enveloppe formée par le WBR
WSI	Wrapper Serial In, port d'entrée sériel de l'enveloppe P1500
WSO	Wrapper Serial Out, port de sortie sériel de l'enveloppe P1500

CHAPITRE 1 INTRODUCTION

Dans le domaine de la microélectronique, la conception de systèmes électroniques, constitués de circuits intégrés, dépend d'une complexité en constante expansion. En effet, les circuits intégrés possèdent de plus en plus de transistors et d'entrées/sorties. Les dernières statistiques, d'après le plan de route ITRS de l'année 2011 (International Technology Road Map for Semiconductors), établissent un nombre entre un et dix milliards de transistors par circuit intégré. Parmi les étapes de conception d'un système électronique, la vérification occupe une place importante. C'est pendant cette phase que le système est exposé à un ensemble de vecteurs de vérification qui permettent de vérifier qu'il fonctionne selon le comportement attendu. Cette phase est possible si et seulement si il existe des modèles pour chacun des circuits intégrés dans le système électronique sous vérification. Or, la difficulté de la vérification dépend à la fois de la complexité et du nombre de circuits intégrés. Plus le nombre d'éléments augmente dans un système électronique, plus la quantité de données requises pour la vérification augmente, tant pour les vecteurs en entrée que les réponses aux sorties. De plus, il est nécessaire d'avoir un accès in-situ pour pouvoir injecter ou observer des signaux afin que les concepteurs puissent vérifier le fonctionnement. En effet, la finesse de gravure et le rétrécissement des contacts des circuits imprimés restreignent un accès direct. Dans le cas de la vérification de circuits intégrés, il est fréquent de retrouver des circuits internes spécifiques aux tests et à la vérification au cœur d'un système, tels que l'utilisation de méthodologies de conception pour le test (DFT, Design For Test [1]) et pour le débogage (DFD, Design For Debug [2]). La première méthodologie influence la construction, l'application et le diagnostic de l'étape de vérification de circuits intégrés tandis que la deuxième permet le débogage dans le cas où des erreurs sont détectées pendant la vérification. Le débogage consiste à l'identification et à la résolution des dysfonctionnements, qui peuvent être introduits par des erreurs de conception ou d'intégration.

Les travaux de ce mémoire se placent dans le contexte du projet de recherche DreamWafer™, dans lequel est développée une plateforme de prototypage rapide de systèmes électroniques, appelée WaferBoard™. Son principe repose sur l'utilisation d'une surface de contacts associée à un réseau d'interconnexions reconfigurables, le WaferIC™. Un système électronique dont les circuits intégrés sont déposés sur le WaferBoard™ peut être directement mis en fonctionnement et son comportement peut être vérifié. Un prototype miniature du WaferIC™, appelé

MiniWaferIC, a été fabriqué puis associé avec un circuit d'alimentation et de configuration. Il est primordial que le fonctionnement du MiniWaferIC soit validé afin de pouvoir poursuivre le développement du WaferBoard™. Cette validation constitue donc une des réalisations présentées dans ce mémoire.

En outre, le WaferBoard™ nécessite des circuits DFT et DFD qui permettront aux utilisateurs de prototyper, donc de vérifier la manière dont fonctionnent leurs systèmes électroniques. Un outil de vérification de systèmes électroniques est présenté dans ce mémoire. Il est adapté pour permettre la capture et la génération de signaux dans un système électronique prototypé sur le WaferBoard™. Contrairement à d'autres dispositifs, comme les analyseurs logiques, qui nécessitent de transporter l'information du point d'extraction vers l'instrument de mesure (ou du générateur au point d'injection) l'outil de vérification est placé in-situ, c'est-à-dire au plus près des interfaces du système. Ceci permet d'augmenter la fréquence de fonctionnement, mais aussi la précision temporelle des mesures en réduisant les délais de propagation.

Dans cette introduction, la vérification de systèmes électroniques sera définie puis confrontée aux procédures de validation et de test. Ensuite, seront décrits les besoins dans le domaine de la vérification des systèmes électroniques. Puis seront listés les différents objectifs de recherche suivis au long de cette maîtrise. Enfin, l'organisation des différents chapitres et du contenu de ce mémoire sera détaillée.

1.1 Vérification, validation et test

Les travaux de ce mémoire discutent de méthodes et d'outils de vérification. Il est donc nécessaire que la vérification soit située puis définie dans le domaine de l'électronique.

Les exigences (*requirements*) d'un système électronique sont généralement interprétées et traduites par le concepteur dans un document appelé « Spécifications » qui contient une description de l'ensemble de ses caractéristiques, performances, fonctionnalités et architectures. L'écriture des spécifications est donc la première étape lors de la conception d'un système électronique. Une fois terminé, le système peut alors être implémenté, c'est-à-dire construit pour satisfaire les spécifications. L'implémentation du système électronique cible correspond à l'utilisation de circuits intégrés disponibles sur le marché et peut mener vers la conception de circuits intégrés dédiés au système électronique, à la configuration de circuits intégrés

programmables et la création de logiciel. Le résultat de l'implémentation est un ensemble de modèles matériels et de descriptions de l'architecture des circuits intégrés et des logiciels mis en jeu. Après avoir été implémentés, les éléments qui composent un système électronique peuvent être fabriqués (s'il y a lieu) puis intégrés. La fabrication est la réalisation physique du ou des circuits imprimés du système électronique et de leur montage et tests. L'intégration implique l'interconnexion des circuits intégrés et des logiciels et environnements nécessaires au fonctionnement du système électronique.

Il existe plusieurs procédures dans la conception de systèmes électroniques qui permettent de s'assurer que la réalisation est conforme aux attentes : la validation, la vérification et le test. Ces trois termes doivent être définis pour être interprétés de la bonne manière dans ce mémoire. En effet, leur définition varie en fonction des auteurs qui les citent dans la littérature et des domaines de l'électronique et de l'informatique. Ainsi est proposée une définition pour la validation, la vérification et le test dans le cadre de la conception de systèmes électroniques. La Figure 1.1 est une illustration de ces procédures à différentes étapes de la conception de systèmes électroniques : les spécifications, l'implémentation puis la fabrication et intégration.

Selon la définition de l'IEEE [3], **la validation vise à confirmer par un examen selon des évidences objectives que les exigences particulières pour une utilisation spécifiques ont été respectées**. La validation interfère avec le client pour lequel le système électronique est réalisé. Son but est de confirmer que les exigences spécifiques du client pour son application sont respectées. Les exigences du client peuvent se classer en termes de fonctionnalités, performances et caractéristiques. Cette confirmation peut s'appliquer à différentes étapes du cycle de conception : de la validation des spécifications jusqu'au produit final. Par exemple, s'il est nécessaire de modifier la spécification à cause d'un imprévu technique lors de la fabrication, la conformité de la modification devra être contrôlée à travers une nouvelle validation.

Selon la définition de l'IEEE [3], la vérification **vise à confirmer par un examen selon des évidences objectives que les exigences spécifiées ont été respectées**. Elle intervient au cours de l'implémentation d'un système électronique, mais aussi après sa fabrication. Le but est alors de s'assurer que les fonctionnalités, les performances et les caractéristiques du système électronique créées avec l'implémentation correspondent à celles qui sont décrites dans les spécifications. Dans le cas de systèmes électroniques, ce type de vérification est assuré par des environnements

de simulation qui utilisent les modèles matériels des circuits intégrés. Des stimulis ou des vecteurs de vérification spécifiques sont appliqués pour observer si le système électronique agit selon les spécifications.

Après la fabrication, la vérification n'intervient plus sur des modèles, mais sur des circuits électroniques réels. Ici, l'objectif est de mesurer les performances et les caractéristiques du système électronique pour pouvoir les confronter aux spécifications. Ces mesures sont par exemple des délais, des taux d'erreurs ou encore des puissances. Il est aussi nécessaire que les fonctionnalités soient contrôlées afin de correspondre au comportement attendu défini dans les spécifications.

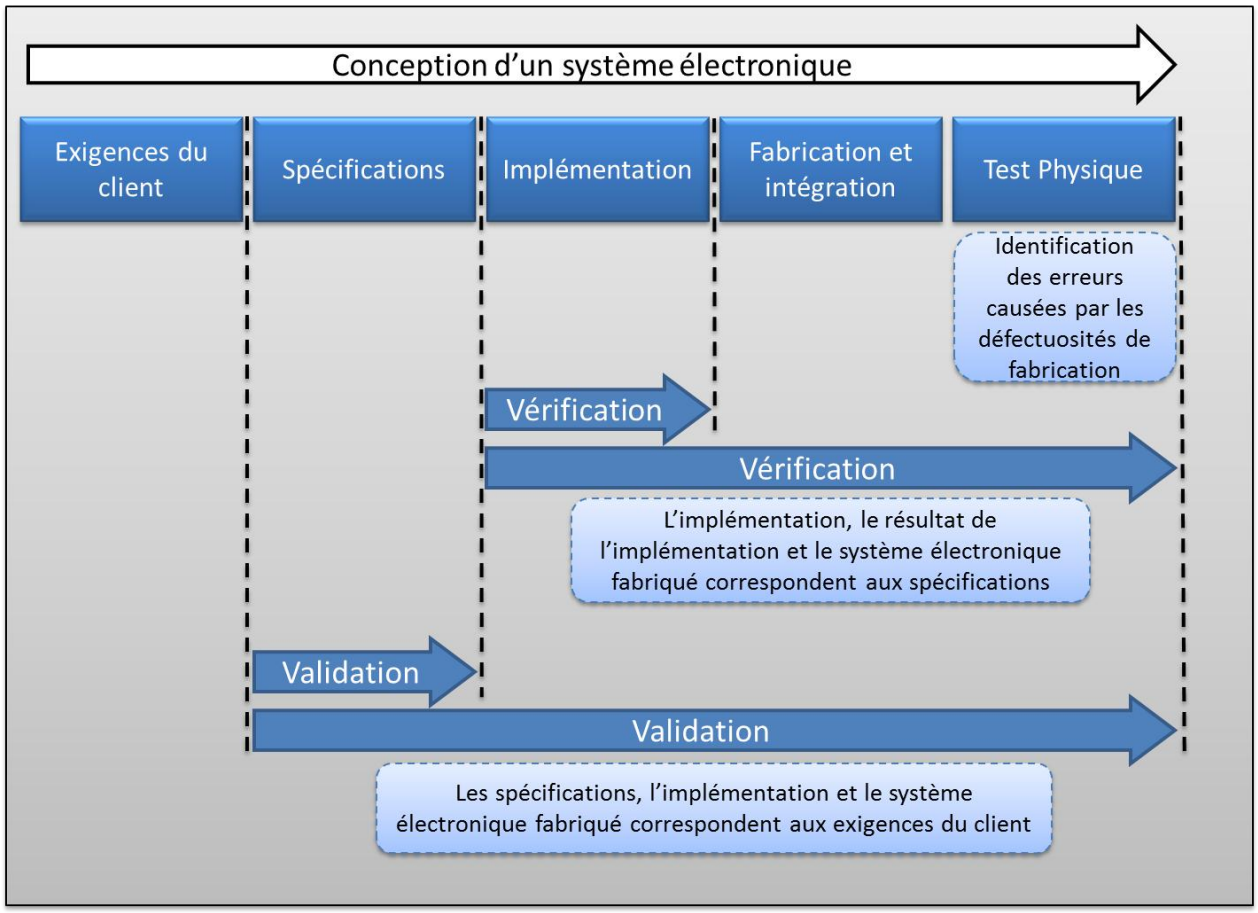


Figure 1.1 : Illustration des procédures permettant de s'assurer qu'un système électronique est conforme aux attentes à plusieurs étapes de sa conception.

Selon le Petit Larousse, le test est la mise à l'essai d'un produit pour vérifier son action, son fonctionnement. Le test est une procédure dont le but est de prouver que le système réalisé est

fonctionnel en cherchant à identifier des erreurs causées par les défectuosités de fabrication. Les examens associés à la vérification et la validation demandent ainsi de créer et d'exécuter une série de tests pour vérifier la conformité. Par l'application de stimulis et de vecteurs de test aux circuits réels, un concepteur peut observer si le système électronique se comporte comme attendu. Une série de tests est généralement définie dans un plan de vérification ou de validation. Dans le cas de dysfonctionnements d'un système électronique, le test peut servir à l'identification des erreurs et des causes, c'est le débogage.

Dans le cadre d'un système électronique, où les circuits intégrés sont nombreux, la vérification peut prendre assez d'ampleur pour devenir l'une des tâches les plus complexes et coûteuses. En effet, pour chaque circuit intégré et leur logiciel (s'il y a lieu), il faut vérifier ses propres fonctionnalités, performances et caractéristiques, mais aussi les interfaces qui existent entre lui et d'autres circuits. De plus, la vérification, et le débogage qui y est associé tendent à prendre de plus en plus de temps tandis que les autres phases de conception s'accélèrent grâce à la réutilisation de circuits ou à l'emploi d'outils de conception automatisée. Il apparaît donc nécessaire de trouver des solutions qui accélèrent ces phases critiques de la conception d'un système électronique.

Dans ce mémoire, on s'intéresse tout particulièrement à la vérification de systèmes électroniques prototypés sur la plateforme WaferBoard™, dans le contexte du projet DreamWafer™, où les circuits intégrés mis en jeu sont des circuits réels (physiques).

1.2 Besoins du domaine de la vérification

Premièrement, dans le domaine de la vérification des systèmes électroniques, de nombreuses entreprises ont recours à des solutions sur mesure qu'ils développent ou achètent pour leurs besoins spécifiques. Un premier besoin dans ce domaine serait de fournir un outil ou une technique de vérification généraliste, adaptable et réutilisable. Il s'agirait d'obtenir un système reconnu et utilisable par tous les concepteurs pour effectuer la vérification de systèmes électroniques.

Deuxièmement, les systèmes sont de plus en plus rapides : pour les systèmes électroniques conventionnels, il n'est plus rare de trouver des fréquences qui dépassent 1 GHz et des interfaces haute vitesse au-delà du 10 Gb/s (OC192, GigE). Ainsi, il serait nécessaire d'améliorer les

techniques de vérification pour permettre une observation et un contrôle des circuits à des fréquences plus élevées.

Troisièmement, les systèmes électroniques sont gravés de plus en plus fin, augmentant donc la densité des circuits et donc le nombre de circuits intégrés du système. Cette augmentation de densité provoque une explosion de la quantité de données qui est produite par les circuits. Or ce sont ces données qui doivent être recueillies ou contrôlées pour effectuer la vérification des systèmes. Il serait donc favorable aux concepteurs d'avoir des méthodes pour gérer d'importantes quantités de données ou pour fragmenter la vérification en plusieurs blocs : c'est la vérification modulaire ou incrémentale.

1.3 Objectifs de recherche

Dans ce mémoire, deux objectifs de recherche principaux se distinguent. Le premier est de proposer un outil de vérification de systèmes électroniques reconfigurables. Ce dernier doit permettre d'interfacer efficacement à un environnement de vérification les différents types de circuits intégrés qui composent un système électronique. De plus, il doit embarquer des modules d'analyse et de génération de signaux numériques nécessaires à la vérification. Enfin, l'outil doit être compatible avec le WaferBoard™ pour constituer une solution de vérification dédiée.

Le deuxième objectif principal est de fournir une preuve de concept du WaferBoard™ à travers des travaux sur le prototype MiniWaferIC. Ce dernier doit d'une part être soumis à des tests électriques pour déterminer les défauts de fabrication. D'autre part, la vérification et le test doivent permettre de confirmer les fonctionnalités et de percevoir les erreurs logiques du prototype.

1.4 Plan du mémoire

Ce mémoire présente le fruit des travaux effectués tout au long de ma maîtrise. Tout d'abord, le Chapitre 2 expliquera en quoi consiste le projet DreamWafer™ et les différentes activités qui y sont reliées, en particulier les étapes pour parvenir à une preuve de concept du WaferBoard™ à partir du MiniWaferIC. Ensuite, dans le Chapitre 3, une revue de littérature sera établie sur les méthodes de vérification et de débogage de systèmes électroniques. Tout en restant généralistes, nous nous centrerons vers les concepts clés qui ont conduit à la mise en œuvre d'une solution.

Puis, le Chapitre 4 proposera l'outil de vérification tel qu'il a été imaginé pour répondre aux objectifs de recherche de notre projet. Il sera présenté ses spécifications, son architecture et comment il permet d'interfacer des circuits intégrés, de générer et d'acquérir des signaux numériques. Par la suite, dans le Chapitre 5, nous fournirons une preuve de concept de notre idée en présentant les activités de test, de vérification et d'évaluation qui ont été réalisées. Enfin, nous résumerons les travaux de recherche de cette maîtrise et y poserons un regard critique. Plusieurs orientations futures seront également proposées.

CHAPITRE 2 PROJET DREAMWAFER™

Le projet de recherche DreamWafer™ a été lancé en 2008 dans le but de proposer une nouvelle solution innovante de prototypage rapide des systèmes électroniques. Fruits des nombreux travaux et publications[1][4][5], il convient d'abord de présenter les résultats principaux à ce jour : le WaferBoard™, le WaferIC™ et le WaferConnect™. Ensuite, les activités qui ont conduit à une preuve de concept du WaferBoard™ à partir du MiniWaferIC sont décrites. Enfin, plusieurs travaux spécifiques sont présentés en annexe comme des contributions personnelles de développement pour le projet DreamWafer™. Ces travaux participent à la constitution des modules nécessaires au fonctionnement de la plateforme WaferBoard™.

2.1 Présentation du projet de recherche DreamWafer™

Le WaferIC™ est un circuit, actif et reconfigurable, intégré à l'échelle d'une tranche de silicium d'un diamètre d'environ 200 mm. Il dispose à sa surface d'une matrice de plots conducteurs, appelés NanoPads, sensibles aux contacts de composants électroniques. Les NanoPads sont configurables en entrées/sorties logiques, alimentation, masse ou nœud flottant. La surface du WaferIC™ remplace les routes de cuivres d'un PCB et permet de relier, d'alimenter ou d'isoler les contacts de circuits selon les besoins du concepteur. Le WaferIC™ est fragmenté en 76 réticules, chaque réticule est composé de 1024 cellules. Enfin, chaque cellule comporte 16 NanoPads, portant le nombre total de ces derniers à 1 245 184. Cette hiérarchie est représentée sur la Figure 2.1. Un réseau d'interconnexions reconfigurable, appelé WaferNet™, permet d'interconnecter les NanoPads entre eux à des distances respectives de 1, 2, 4, 8, 16 et 32 cellules, ceci dans les directions nord, sud, est et ouest.

Le WaferBoard™ est la structure matérielle complète qui permet d'intégrer le WaferIC™ (Figure 2.2). Il contient des circuits dédiés à l'alimentation, à la communication avec l'extérieur, mais aussi des structures mécaniques pour assurer un bon contact des composants (couvercle) ou un refroidissement efficace de tout le système (radiateurs, ventilateurs).

---

[1] Les publications initiales sont dans notre liste de référence, la totalité est disponible sur le site internet officiel à l'adresse : http://www.DreaMiniWaferICaferTM.com/research.html

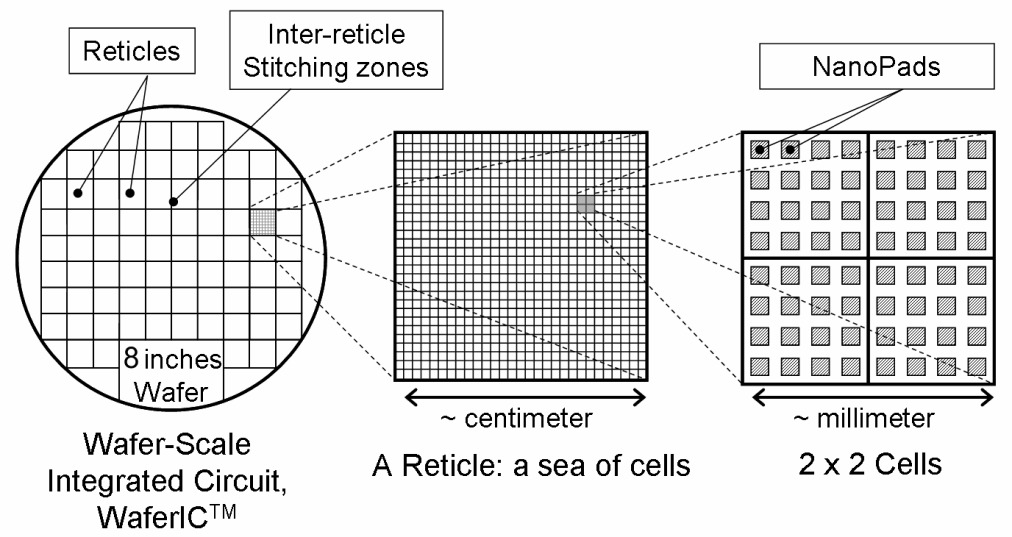


Figure 2.1: Hiérarchie du WaferIC™ à différentes échelles (tirée de [5]).

Figure 2.2 : Schéma 3D du WaferBoard™, image tirée de WaferBoard™ Product Brief sur www.dreamwafer.com.

Pour réaliser le prototype d'un système électronique, un usager dépose les composants qu'il désire sur la surface utile du WaferIC™ puis referme le couvercle du WaferBoard™ pour appliquer une pression et assurer un contact électrique avec les pins des circuits.

Enfin, le contrôle du WaferBoard™ et du WaferIC™ est assuré par un logiciel : WaferConnect™. Ce dernier procure une interface graphique permettant à l'usager de reconnaître et visualiser les composants déposés sur le WaferIC™ puis de créer dynamiquement des routes à travers le design. La Figure 2.3 montre une capture de la représentation logicielle des composants et des routes sur le WaferIC™.

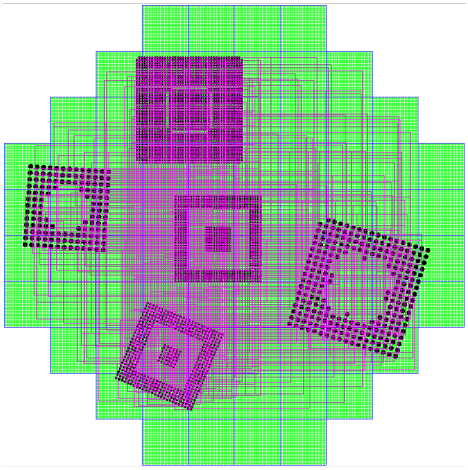


Figure 2.3 : Représentation logicielle avec WaferConnect™ du WaferIC™ et de différents composants et routes du système en cours de prototypage. Image obtenue par capture d'écran du logiciel WaferConnect™.

2.2 Vérification du MiniWaferIC

Le WaferIC™ a été développé de manière incrémentale, si bien que plusieurs versions de prototype ont vu le jour. Au cours de l'automne 2012, le prototype de la version 3 a été reçu. Ce prototype est une version condensée du WaferIC™ réel. Appelé MiniWaferIC, il contient 4 images de réticules plutôt que 76[2]. Les tests du MiniWaferIC ont été arrêtés pour des raisons organisationnelles à la fin de l'automne 2012. Vers la fin de la session suivante (hiver 2013), une nouvelle équipe[3] a été créée, chargée de reprendre et poursuivre les tests sur le PowerBlock et le MiniWaferIC. L'objectif était de valider le fonctionnement prévu par le design du WaferIC™ et

[2] Un réticule est un module unitaire du plus haut niveau dans la logique du WaferIC™.

[3] L'équipe de test du MINIWAFERIC et PowerBlock été constituée de 3 personnes, Gontran Sion, Safa Berrima et Sylvain Charasse. La personne qui était originalement responsable des tests était Oussama Berriah.

du PowerBlock. Le PowerBlock est un circuit dédié à l'alimentation et à la communication d'une taille équivalente à 2×2 images de réticule du WaferIC™. Le schématique du PowerBlock est donné en annexe (Figure A.VI).

Dans cette partie sera d'abord présenté l'environnement de test mis en œuvre pour le test du MiniWaferIC et du PowerBlock. Puis seront décrits les tests effectués avec leurs résultats. Enfin, les contributions personnelles apportées lors des activités de test seront abordées.

2.2.1 Environnement de test

La Figure 2.4 donne un aperçu simplifié du circuit combinant le PowerBlock et le MiniWaferIC.

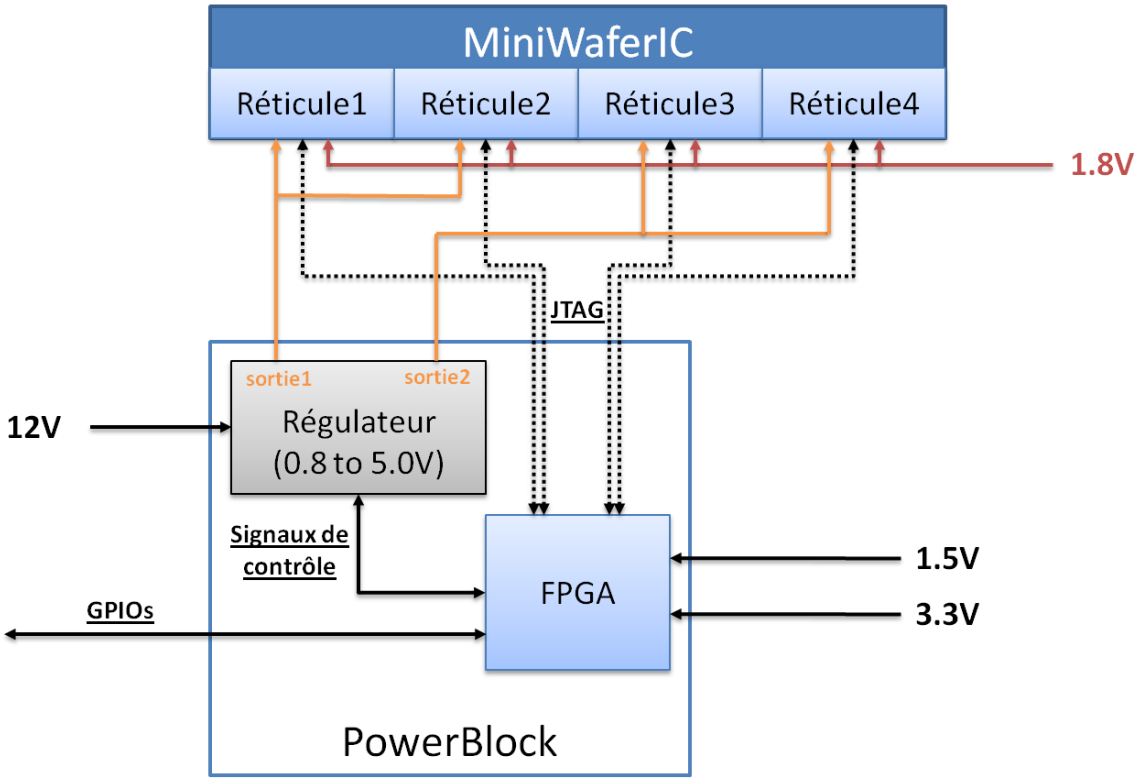


Figure 2.4 : Schéma simplifié du circuit sous test, PowerBlock et MiniWaferIC.

Le PowerBlock contient essentiellement un FPGA de contrôle (Actel IGLOO AGL060V5) et un régulateur de tension (IP1202). Le FPGA contrôle le fonctionnement du régulateur de tension et possède un bus de communication JTAG avec chacun des quatre réticules du MiniWaferIC. Le régulateur est contrôlé par la sélection d'une tension de sortie et l'activation ou non d'une ou de deux de ses sorties.

Le bus de communication utilisé pour communiquer avec l'extérieur est constitué de 6 GPIOs (General Purpose Input Output). Ces GPIOs sont normalement connectés au Bottom PCB (circuit de gestion des communications), mais pour les tests de validation ils sont utilisés de plusieurs façons en fonction des besoins. Il existe notamment une connexion directe avec un microcontrôleur, lui-même interfacé à Matlab sur un ordinateur de test.

Le circuit nécessite quatre alimentations indépendantes :

- 1.5V : Alimentation du cœur logique du FPGA
- 3.3V : Alimentation des deux banques d'entrées et sorties du FPGA
- 1.8V : Alimentation de la circuiterie 1.8 V dans le WaferIC
- 12V : Alimentation du régulateur de tension

La Figure 2.5 est une photo avec et sans câblage du PCB du PowerBlock vu du dessus. Le MiniWaferIC se trouve en dessous.

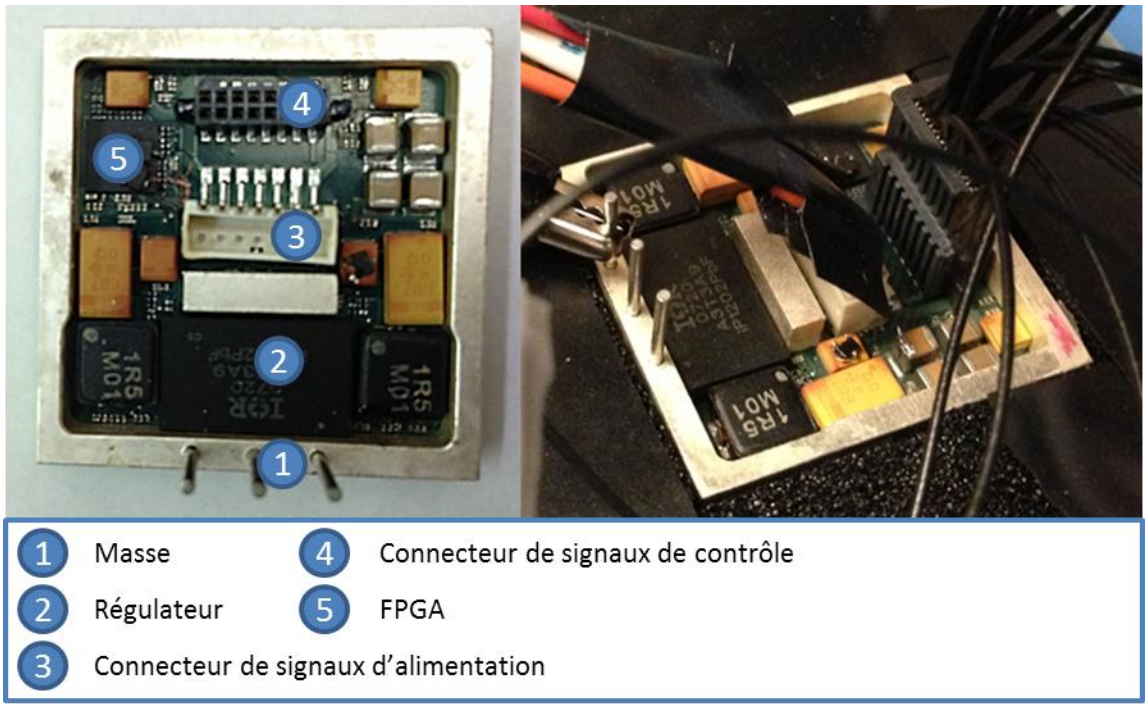


Figure 2.5 : Photo du PowerBlock vu de dessus.

2.2.2 Tests et résultats

Les tests ont été divisés en trois étapes principales[4] :

- Validation du fonctionnement du FPGA du PowerBlock : vérifier que le FPGA est correctement alimenté, que sa logique fonctionne et que les entrées/sorties sont fonctionnelles;
- Validation de la bonne alimentation du MiniWaferIC : vérifier que le régulateur de tension fonctionne, s'assurer d'avoir une consommation de courant pertinente;
- Validation du fonctionnement logique du MiniWaferIC : vérifier les fonctionnalités attendues du MiniWaferIC en fonction du contrôle par le bus JTAG.

2.2.2.1 Fonctionnement du FPGA dans le PowerBlock

Pour valider le fonctionnement du FPGA, une série de tests ont été effectués en utilisant les GPIOs pour établir la communication entre le FPGA et l'ordinateur. La liaison avec le FPGA a été construite avec un programme Matlab permettant de communiquer en USB et un convertisseur USB à UART (UM232R). Le schéma de communication est présenté sur la Figure 2.6. Le protocole UART a été choisi pour sa simplicité et sa rapidité d'implémentation.

Figure 2.6 : Chemin de communication entre l'ordinateur de test et le FPGA du PowerBlock

Du côté du FPGA, un simple module UART permet de recevoir et d'envoyer des paquets. Une machine à états traite les paquets pour extraire les informations transmises ou reçues, principalement des commandes pour activer différentes fonctions. L'horloge du FPGA est

[4] Les tests devaient être méticuleux, car il n'y avait que deux prototypes de disponibles. Un de ces prototypes a notamment cessé de fonctionner pendant les tests. De plus, les tests sont présentés de manière synthétique pour une compréhension rapide des travaux réalisés. Le rapport technique « Test du MiniWaferIC.docx » donne des informations plus détaillées sur les tests. Il existe aussi un journal de bord des tests : « journal de bord.docx ».

générée à l'aide d'un générateur de fonction externe et acheminée jusqu'à lui par une entrée GPIO.

Les tests effectués, et vérifiés fonctionnels, ont été :

- Loop-back : le FPGA réceptionne des paquets UART puis les renvoie tels quels à Matlab.
- Contrôle de signaux JTAG et de signaux de contrôle du régulateur de tension : le FPGA envoie des signaux sur ses différentes sorties en fonction des paquets reçus sur l'UART.

Cependant, un GPIO (« CLK1 ») était non fonctionnel sur le PowerBlock, laissant à 5 le nombre de GPIOs utilisables.

2.2.2.2 Alimentation du MiniWaferIC

Chaque réticule du MiniWaferIC contient deux plans d'alimentation : 3.3 V et 1.8 V. La séquence de mise sous tension du MiniWaferIC se déroule en deux phases. D'abord, le régulateur de tension doit être correctement configuré (à partir des signaux de contrôle du FPGA) pour fournir une tension d'au minimum 1.8 V au plan 3.3 V des réticules. Ensuite seulement, on doit alimenter progressivement le plan 1.8 V à 1.8 V. Ce protocole doit être respecté, car il existe des jonctions *pn* entre les deux plans d'alimentation qui se doivent d'être polarisées en inverse. Or si le plan d'alimentation 3.3 V se trouvait à 0 V et que le plan 1.8 V était actif, la jonction *pn* entre les deux plans serait polarisée en direct. Ceci aurait pour effet de créer une importante consommation de courant dans le MiniWaferIC et pourrait l'endommager.

Néanmoins, avant d'alimenter le plan 1.8 V, plusieurs seuils de tension ont été testés pour vérifier la précision du régulateur, soient les paliers 1.2 V, 1.5 V, 1.8 V, 2.0 V et 3.3 V. La tension des sorties du régulateur est spécifiée à 3.3 V pour un fonctionnement normal. Pour sélectionner la tension de sortie du régulateur, des sorties du FPGA sont reliées à un diviseur de tension (un pour chaque sortie du régulateur[5] (voir le schématique du PowerBlock, Figure A.VI).

Toutes les observations concernant le régulateur ont donné des résultats favorables sauf le signal de statut « PGOOD » du régulateur qui ne semble pas fonctionner tel que spécifié ou non

[5] Schématique du Powerblock, chemin sur SVN :
resmiq.info.uqam.ca/local/SVN/DreamWafer/trunk/MW_Test/pertinent/0 PowerBlocks schematic_SK3914_1_oct14 V36.pdf

observable sur un pin du FPGA. Ce signal permet normalement de savoir si les sorties du régulateur sont actives.

Concernant les consommations statiques de courant, nous avons observé 0.42 A sur l'alimentation 12 V des régulateurs et 3.96 A sur l'alimentation 1.8 V soit une puissance totale en entrée d'environ 12 W. Il est à noter que les signaux TRST des bus JTAG vers les quatre réticules sont maintenus à '0' pour réinitialiser le réseau JTAG dans le MiniWaferIC. Sans ce dernier détail, la consommation de courant sur le plan 12 V augmente significativement (environ 0.7 A pour le cas observé, soit 66 % de plus). Le signal TRST initialise toutes les bascules de la chaîne JTAG dans le WaferIC. Ces bascules déterminent l'état de la circuiterie. La surconsommation peut provenir de plusieurs sources, dont des courts circuits au niveau des circuits d'entrées-sorties des NanoPads et principalement des régulateurs distribués.

2.2.2.3 Fonctionnement logique du MiniWaferIC

La validation du fonctionnement logique du MiniWaferIC consiste à s'assurer que le prototype soit contrôlable à partir des bus JTAG et que les fonctionnalités soient opérationnelles. Cependant, il est d'abord nécessaire de vérifier que la configuration du réseau JTAG du MiniWaferIC est fonctionnelle. En effet, les réticules, entités du plus haut niveau du MiniWaferIC, sont constitués de cellules logiques parcourues par un réseau JTAG reconfigurable. La méthode utilisée consiste d'abord à générer les signaux JTAG qui permettent de créer un chemin (Figure 2.7) de l'entrée JTAG (TDI) à une cellule donnée du MiniWaferIC puis jusqu'à la sortie JTAG (TDO). Ensuite, une signature binaire est envoyée en entrée du réseau JTAG. Son observation en sortie confirme qu'un chemin a été construit entre l'entrée et la sortie JTAG du MiniWaferIC.

La Figure 2.8 est un exemple de la méthode et montre les cinq signaux JTAG qui sont mis en jeu. Le signal TCK est l'horloge, il reste actif tout au long de la séquence. TMS et TDI sont les signaux de contrôle qui permettent de contrôler le réseau JTAG pour configurer et créer un chemin au travers des cellules du MiniWaferIC. TDO est le signal de sortie JTAG du MiniWaferIC, on peut y observer un signal à la fin de la séquence. Ce dernier correspond à la signature binaire qui confirme qu'un chemin a été créé entre l'entrée et la sortie JTAG du MiniWaferIC.

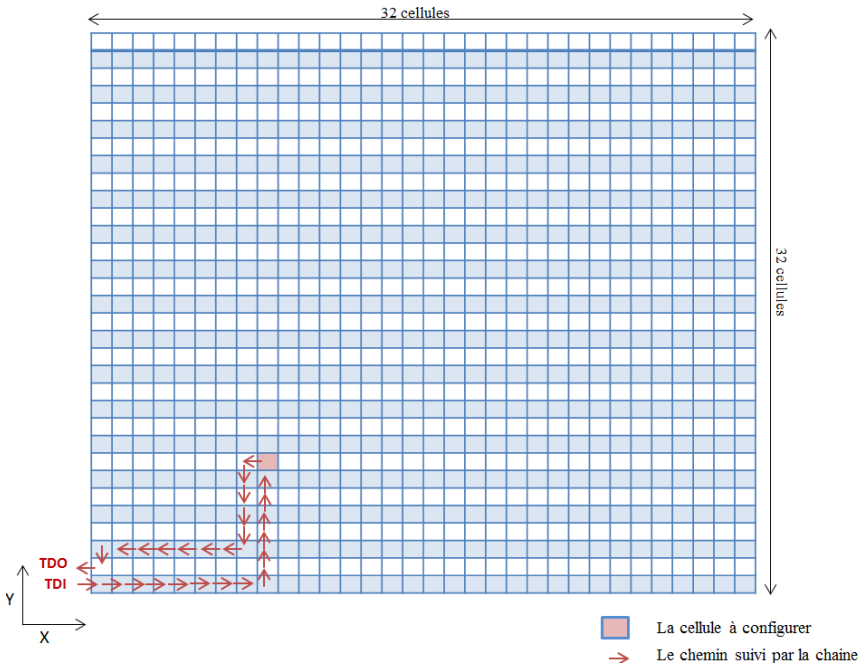


Figure 2.7 : Exemple d'un chemin JTAG reliant TDI et TDO en passant par une cellule particulière d'un réticule. Image tirée du rapport de test du MiniWaferIC. Chemin sur SVN : resmiq.info.uqam.ca/local/SVN/DreamWafer/trunk/MW_test/Rapports.

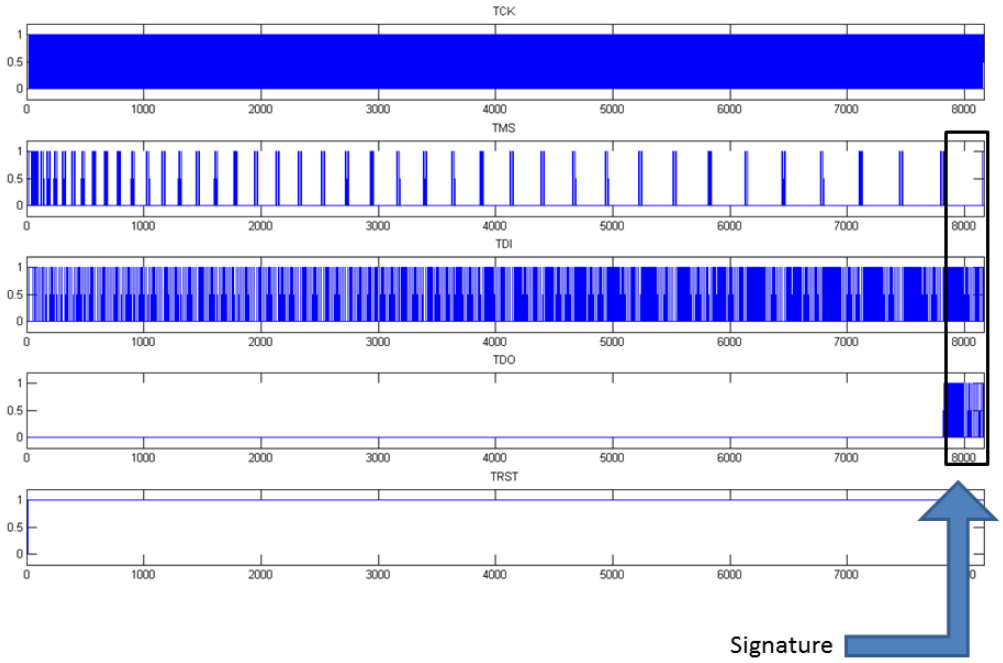


Figure 2.8 : Observation de la signature en sortie du réseau JTAG, après configuration d'un chemin donné. Image tirée du rapport de test du MiniWaferIC. Chemin sur SVN : resmiq.info.uqam.ca/local/SVN/DreamWafer/trunk/MW_test/Rapports.

Un ensemble de chemins différents ont été vérifiés, notamment le chemin le plus court et un chemin qui va jusqu'au centre du MiniWaferIC. Étant tous fonctionnels, les travaux actuels se dirigent vers l'écriture d'un script qui permettra de vérifier tous les chemins JTAG possibles dans le MiniWaferIC.

Par la suite, il sera nécessaire de vérifier les autres fonctionnalités du MiniWaferIC. Le réseau JTAG sera alors utilisé pour configurer différentes cellules d'un réticule pour des tests spécifiques. Un schéma est donné, représentant la manière dont sont générées les trames de bits JTAG pour configurer les réticules du MiniWaferIC (Figure 2.9).

Tout d'abord, la fonctionnalité désirée du MiniWaferIC est sélectionnée. Il peut s'agir par exemple de vouloir configurer un contact donné du WaferIC pour qu'il alimente une entrée d'un circuit intégré. Le générateur de trames de bits JTAG, développé dans le logiciel WaferConnect™, est alors utilisé pour créer une trame de configuration JTAG correspondante. Ensuite, une simulation ModelSim est effectuée avec le modèle matériel (décrit en VHDL et SystemC) du WaferIC™ et les trames JTAG générées comme entrée. Cette étape permet de vérifier que la fonctionnalité est opérationnelle en simulation. Puis les trames JTAG sont transmises au FPGA du PowerBlock, encapsulées dans des paquets USB. Le FPGA les récupère enfin dans les paquets USB pour les transmettre aux réticules et effectuer le test désiré.

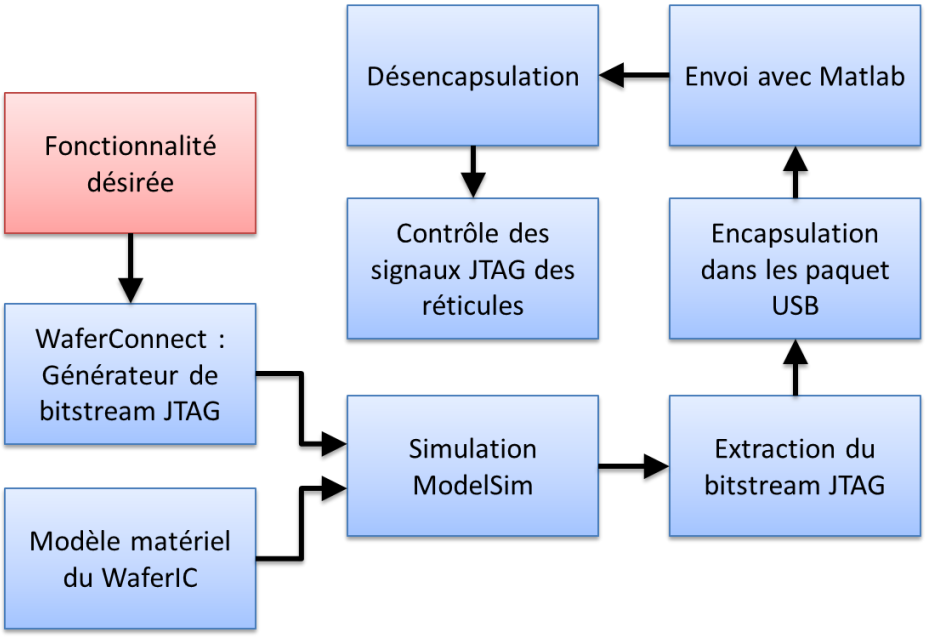


Figure 2.9 : Flot de génération des signaux de contrôle JTAG pour les réticules du MiniWaferIC.

2.2.2.4 Caractérisation en température et refroidissement du prototype

Dans la section 2.2.2.2 Alimentation du MiniWaferIC page 14, il a été montré que la consommation statique de notre prototype, sous tension est d'environ 12 W (PowerBlock et MiniWaferIC). Cette consommation apporte une quantité de chaleur importante, car le PCB est petit et la dissipation thermique y est presque inexistante. Or les circuits électroniques subissent un vieillissement accéléré s'ils sont exposés à de plus hautes températures, et seul un prototype fonctionnel est actuellement disponible pour mener les expériences. Dans un objectif d'accroître la durée de vie du PowerBlock et du MiniWaferIC, une solution d'extraction de la chaleur est proposée dans cette partie. Ainsi a été mise en place une caractérisation en température du prototype sous tension[6] associé à un refroidissement à base de ventilateurs. Cette expérience a pour objectif de démontrer que les tests du MiniWaferIC peuvent être effectués à une température acceptable dans le cas où un dispositif d'extraction de la chaleur est utilisé.

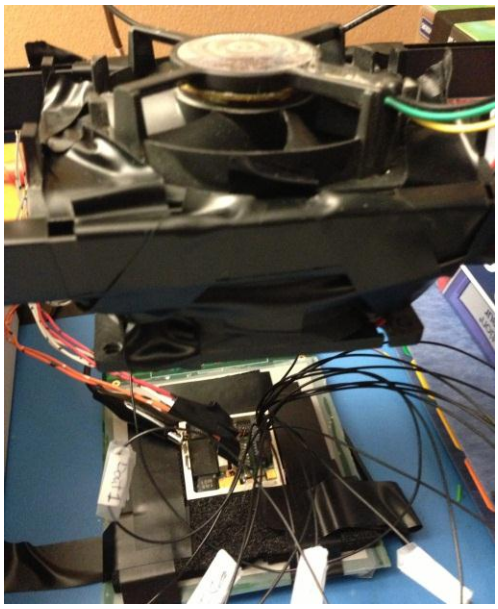


Figure 2.10 : Structure de refroidissement pour les tests du MiniWaferIC.

La Figure 2.10 montre la structure de refroidissement qui a été réalisée en utilisant un ventilateur de processeur et un ventilateur de boîtier d'ordinateur. L'alimentation des ventilateurs est de

[6] Les résultats complets de cette expérience sont résumés dans le document « Caractérisation en température du MINIWAFERIC.xslx ».

12 V. Comme cette solution est simple, deux ventilateurs sont utilisés plutôt qu'un pour augmenter la fiabilité du refroidissement. Si l'un des ventilateurs cesse de fonctionner, l'autre continuera d'assurer l'extraction de chaleur.

Les résultats de la caractérisation en température sont résumés dans la Figure 2.11. Pendant l'expérience, la température ambiante était de 22.4 °C. Les deux courbes représentent la température mesurée à la surface du régulateur de tension et à la surface du FPGA sur le PCB du PowerBlock. La température a été mesurée avec un thermomètre infrarouge Extech 42509.

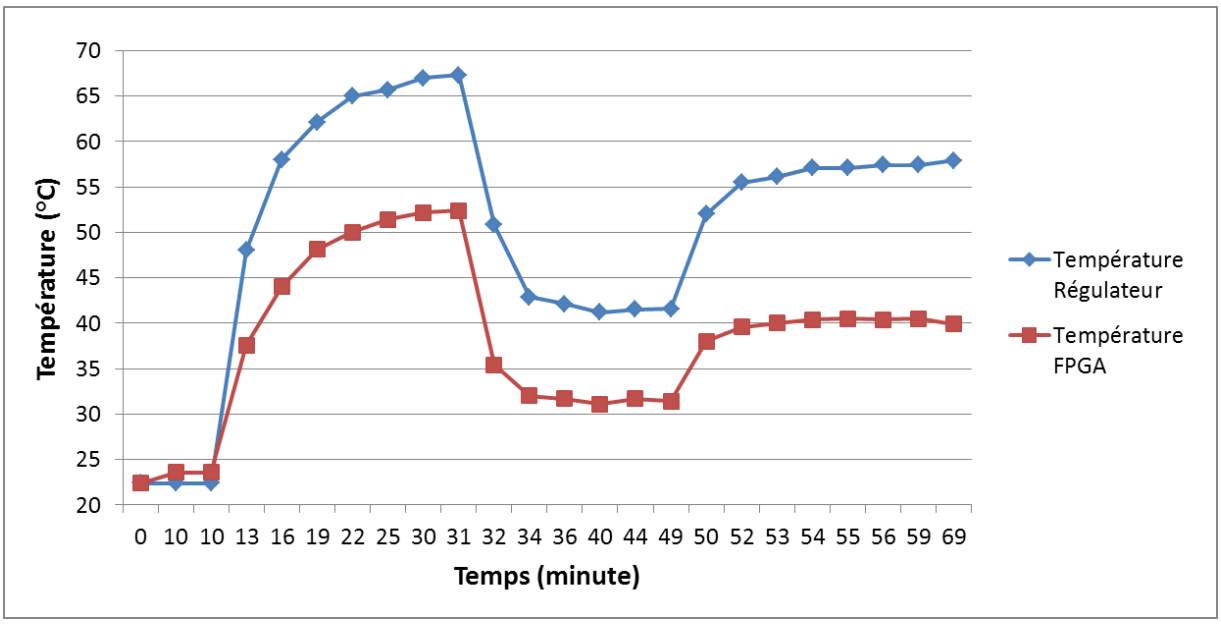


Figure 2.11 : Courbes de température obtenues avec la caractérisation en température du MiniWaferIC sous tension.

Pendant les 10 premières minutes, seul le FPGA était alimenté avec ses deux alimentations (1.5 V et 3.3 V). Ensuite, trois paliers sont identifiés, correspondant à trois évènements particuliers de l'expérience. Entre chaque évènement, la température a été stabilisée. À 10 minutes, le régulateur du PowerBlock a été activé et configuré à 3.3 V, sa température monte alors jusqu'à presque 70 °C (régulateur). À 31 minutes, les ventilateurs sont mis sous tension et la température se stabilise autour de 42 °C (régulateur) en moins de 10 minutes. Cet essai confirme que les ventilateurs offrent un refroidissement efficace et rapide dans le cas où ils sont utilisés avec notre prototype. Enfin, à 49 minutes, le plan d'alimentation 1.8 V du MiniWaferIC est activé, où le PowerBlock et le MiniWaferIC sont dans un mode d'alimentation fonctionnel. La température se stabilise finalement autour de 56 °C (régulateur) et 40 °C (FPGA).

En conclusion, l'utilisation de ventilateurs est pertinente pour effectuer des tests de longue durée puisque le point observable le plus chaud ne dépasse pas 60 °C. À titre de comparaison, un processeur de Intel fabriqué dans la même technologie, 180 nm, affiche une température maximale de 73 °C à l'extérieur de son boîtier [6]. Le fait de diminuer la température de l'unique prototype pour effectuer des tests une sécurité supplémentaire pour ne pas altérer son fonctionnement. Cependant, il est important de savoir que dans le WaferBoard, des structures d'évacuation de la chaleur sont prévues pour le refroidissement du WaferIC et des 21 PowerBlocks qui s'y trouveront.

CHAPITRE 3 MÉTHODES DE VÉRIFICATION

La vérification de systèmes électroniques consiste à observer le comportement du matériel et du logiciel pour pouvoir observer si les fonctionnalités, les performances et les caractéristiques correspondent aux attentes des spécifications prévues lors de la mise en fonction ou dans différentes situations fonctionnelles. La durée de cette phase de conception des systèmes est dominante dans le cycle de conception, car sa complexité croît avec le nombre d'éléments et de fonctions qui composent le système. La loi de Moore se perpétue encore aujourd'hui, signe que la complexité fonctionnelle des systèmes augmente toujours plus. Ainsi, la vérification est vouée à prendre une plus grande part du processus de conception. Les objectifs et intérêts de ce domaine sont donc de trouver des méthodes rapides, précises et réutilisables afin de limiter l'impact de la vérification sur le temps de conception des systèmes électroniques.

Dans ce chapitre, une revue des méthodes de vérification utilisées par les concepteurs est présentée. Plus particulièrement, cette revue est orientée vers la vérification des circuits numériques, les aspects électriques, électromagnétiques et analogiques ne sont pas abordés. Ensuite, les avantages d'utiliser des solutions de prototypage matériel et reconfigurable seront répertoriés. Puis une méthodologie de vérification compatible avec le projet DreamWafer™ sera détaillée. Enfin, une norme, l'IEEE 1500 (ou P1500) qui s'inscrit comme un moyen proposé dans ce mémoire de développer des solutions de vérification généralistes et réutilisables est introduite.

3.1 Vérification par simulation logicielle

Dans de nombreux cas de conception, la première étape de la vérification d'un système électronique consiste à vérifier les fonctionnalités logiques du système. Les circuits intégrés qui le composent sont généralement décrits et modélisés en langage de description matérielle (HDL : Hardware Description Language) afin d'être simulés sur un logiciel.

Il existe différents langages HDL qui permettent de simuler le comportement à des niveaux d'abstractions différents. Par exemple, le VHDL ou le Verilog permettent de décrire des circuits logiques au niveau transfert de registre (RTL) ou de manière plus précise à l'échelle de la porte logique tandis que le System Verilog est plutôt adapté à la construction de structures complexe de vérification ou à l'intégration de larges systèmes [7]. De plus, des langages ou des librairies de méthodologie de vérification ont été développés pour faciliter la vérification à partir des

simulateurs logiciels en fournissant des modules et des fonctions réutilisables. Par exemple, le langage de vérification «e» offert dans la suite Cadence [8] bonifie les langages VHDL ou Verilog. Pour le langage System Verilog, il existe la librairie de vérification standard UVM (Universal Verification Methodology) [9], basée sur la librairie open source OVM (Open Verification Methodology) [10] et qui offre des fonctionnalités semblables à celles de «e». Parmi les outils présents dans ces librairies, on trouve notamment des observateurs qui effectuent les vérifications unitaires automatiquement en se basant sur des assertions (tel que PSL [11], Property Specification Language). Ces dernières sont des expressions relatives au comportement du système qui sont vérifiées de manière automatique au cours de la simulation. Les assertions accélèrent la vérification [12] de manière significative quand le nombre de vérifications unitaires se compte en milliers.

En outre, il est possible d'introduire des considérations temporelles dans les simulations logiques à partir de modèles de délai aux différentes fonctions logiques. Cette option permet de résoudre des erreurs temporelles qui sont invisibles lors d'une simulation logique sans délai. On notera que les concepteurs tentent d'éviter de faire ces types de simulation qui sont voraces en temps de calcul.

Cependant, devant la complexité des systèmes électroniques développés aujourd'hui, les temps de simulation sont très longs, notamment lorsqu'il faut vérifier une fonctionnalité dont la durée réelle est très grande devant la période d'horloge. Cette limite est suffisamment contraignante pour que plusieurs autres techniques de vérification aient vu le jour pour réduire la durée de la phase de vérification des systèmes.

3.2 Vérification avec FPGA

Les FPGA (Field Programmable Gates Array) permettent d'implémenter en matériel des circuits décrits en langage HDL. Ils permettent d'obtenir une version fonctionnelle de circuits logiques et peuvent imiter le fonctionnement du système cible jusqu'à une fréquence de plusieurs centaines de MHz.

3.2.1 Prototypage

Les FPGA sont souvent utilisés avec leur carte de développement pour la vérification [13]. Des exemples de cartes de dernières générations et leurs caractéristiques sont présentées au Tableau 3.1 :

Tableau 3.1 : Exemples de cartes de développement FPGA de dernière génération et leurs caractéristiques. Sources : www.xilinx.com, www.altera.com, www.microsemi.com.

Carte de développement et FPGA	**Caractéristiques du FPGA**	**Caractéristiques de la carte**
Xilinx Virtex-7 FPGA VC707 Evaluation Kit *FPGA :* Xilinx Virtex XC7VX485T	485 760 cellules logiques 700 entrées/sorties 37 Mbits de mémoire interne	Configuration JTAG, SPI Flash ou BPI Flash PCI Express Mémoire DDR3 1 GB Ports de communication : PCI Express, USB 2.0, GTX, SFP+ Port HDMI 2 connecteurs d'expansion, 276 connexions au total
DSP Development Kit, Stratix V Edition *FPGA :* Stratix V GS 5SGSMD5N	457 000 cellules logiques 864 entrées/sorties 39 Mbits de mémoire interne	Configuration JTAG ou Flash Fast Passive Parallel (FPP) Mémoire DDR3 1.152 GB Ports de communication : PCI Express, USB 2.0, Ethernet, SMB 2 connecteurs d'expansion, 168 connexions au total
SmartFusion2 Development Kit *FPGA :* SmartFusion 2 M2S050T	50 000 cellules logiques 1.5 Mbits de mémoire interne	Configuration JTAG ou SPI Flash Mémoire DDR3 0.512 GB Ports de communication : PCI Express, USB 2.0, Ethernet, SMB, I2C, SPI, UART Convertisseur analogique numérique

L'avantage de ces cartes de développement est qu'elles forment un environnement de développement adaptable à différentes utilisations et réutilisable pour plusieurs designs. Ce sont des stations de prototypage où les descriptions HDL de systèmes électroniques sont émulées plutôt que simulées, c'est-à-dire que les circuits fonctionnent réellement sur un support matériel. Ainsi, la complexité n'a plus lieu d'être virtualisée et traitée par un ordinateur. Cependant, il est nécessaire d'instrumenter ces cartes de développement pour obtenir l'observabilité et la

contrôlabilité nécessaires à la vérification du circuit testé. Cette tâche est décrite dans la partie suivante.

3.2.2 Techniques de vérification sur FPGA

Pour permettre la vérification d'un système implémenté sur un FPGA, il existe deux principes de base. Le premier consiste à rediriger les signaux d'intérêt vers les entrées/sorties du FPGA puis les connecter à un analyseur logique et à un générateur de vecteurs externe. Le deuxième est l'implémentation de modules de vérification dédiés à l'intérieur du FPGA et l'utilisation de mémoire interne [13]. Ces techniques sont brièvement décrites dans les prochaines sous-sections. Finalement, il est courant de voir des approches mixtes qui utilisent les deux principes.

3.2.2.1 Analyseurs logiques externes

Un analyseur logique permet d'échantillonner des signaux numériques comme le ferait un oscilloscope pour des signaux analogiques. Ces outils sont faits pour avoir une mémoire et un nombre important de canaux. Ils disposent de fonctionnalités de déclenchement complexe et d'analyse et de comparaison des signaux.

Les caractéristiques techniques des analyseurs logiques sont très variables, tout comme leur prix. Il y a d'abord des versions bon marché qui se connectent à un ordinateur, la fréquence d'acquisition se limite alors souvent à quelques mégahertz; (voir exemple sur la Figure 3.2). De plus, le nombre de canaux est souvent limité. Ensuite, il y a des modules indépendants comme le sont la plupart des oscilloscopes. Il y a alors des versions nettement plus dispendieuses et plus complexes, pour une plus grande fréquence d'acquisition (jusqu'à plusieurs Gigahertz) ou pour plus de canaux et d'espace mémoire.

Les Figure 3.1 et Figure 3.2 donnent des exemples de deux types de version d'analyseur logique.

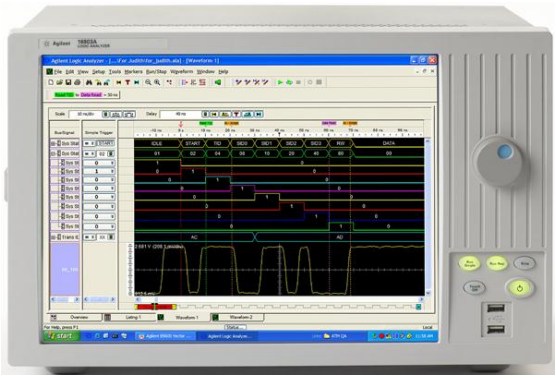


Figure 3.1 : Analyseur logique indépendant : 16802A de Agilent — 68 canaux à une profondeur de 32Mbits, fréquence entre 0.5 Hz et 4 GHz — environ 15 000 $ (ce coût ne prend pas en compte les sondes actives qui permettent de relier le système à l'analyseur sans compromettre l'intégrité des signaux, environ 4000 $ pour une sonde valable jusqu'à 4 GHz).

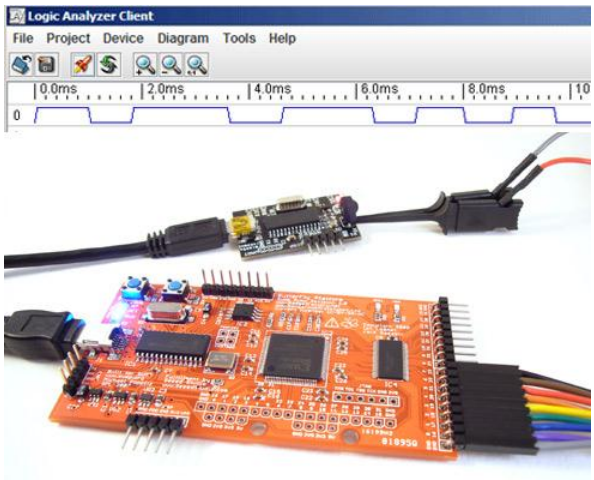


Figure 3.2 : Analyseur logique avec interface par ordinateur : Open Bench Logic Sniffer — 32 canaux à une profondeur de 6Kbits, fréquence entre 50 MHz et 100 MHz — environ 50 $.

Cependant, pour la vérification de circuits nécessitant l'observation de nombreux signaux internes, le nombre d'entrées/sorties disponibles peut faire défaut. Ainsi, même un analyseur disposant de nombreux canaux ne suffira pas à observer un nombre de signaux suffisant pour une vérification de qualité. Il peut alors devenir pertinent de faire appel à des solutions de vérification dédiées.

3.2.2.2 Modules de vérification dédiés

Les modules de vérification dédiés sont des solutions intégrées au design cible. Le but principal est de pouvoir observer les signaux internes des circuits sans avoir à faire appel à des entrées/sorties supplémentaires sur les FPGA. Ce module devient alors une composante intégrée au système vérifié dans le FPGA. Il représente un coût en surface non négligeable, mais il donne un accès aux signaux, permettant d'observer des erreurs et donc réduire le temps nécessaire à la validation.

Un exemple connu de ce genre de module est l'outil Chipscope Pro de Xilinx [14], spécialement conçu pour la vérification et le débogage des FPGA de Xilinx (Figure 3.3).

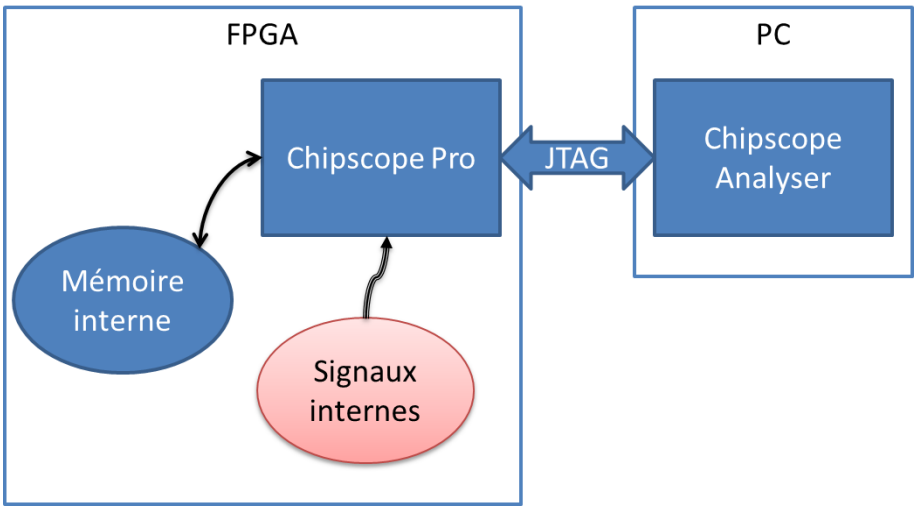


Figure 3.3 : Schéma de fonctionnement du module Chipscope Pro de Xilinx.

Ce module utilise la connexion JTAG entre le FPGA et un ordinateur avec un logiciel de vérification pour capter les signaux observés dans le FPGA. Ainsi, il n'y a pas besoin d'entrées/sorties réservées pour la vérification. Chipscope Pro utilise les unités de mémoire interne des FPGA, les BRAM (Block RAM), pour mémoriser les signaux après ou avant un signal de déclenchement. Les avantages de Chipscope sont de pouvoir être synchronisé à l'horloge du système et de pouvoir intégrer les circuits de vérification directement en VHDL dans un design. Les principaux inconvénients de ces outils sont que les BRAM des FPGA sont limités en espace de stockage, 2.1 Mbits pour le Spartan6 LX45. Également, les ressources de logique et de routage requises pour les modules de commande et la redirection des signaux peuvent être insuffisantes.

Le principal concurrent de Xilinx, Altera, fournit un outil similaire à Chipscope pour ses propres FPGA : SignalTap II [15].

3.2.2.3 Mélange de solutions

Pour pallier le manque de mémoire introduit par Chipscope Pro, Xilinx s'est allié avec Agilent pour proposer une solution pertinente qui tire avantage des solutions intégrées et des analyseurs logiques. Un circuit logique appelé ATC2 (Agilent Trace Core 2 [14]) a été conçu pour se placer entre l'interface de contrôle de Chipscope Pro et un analyseur logique externe Agilent (Figure 3.4). Ce circuit réduit dans tous les cas le nombre d'entrées/sorties du FPGA nécessaires vers l'analyseur en fonction du nombre de signaux observés simultanément. Il est possible d'utiliser entre 4 et 64 entrées/sorties du FPGA en fonction des performances désirées.

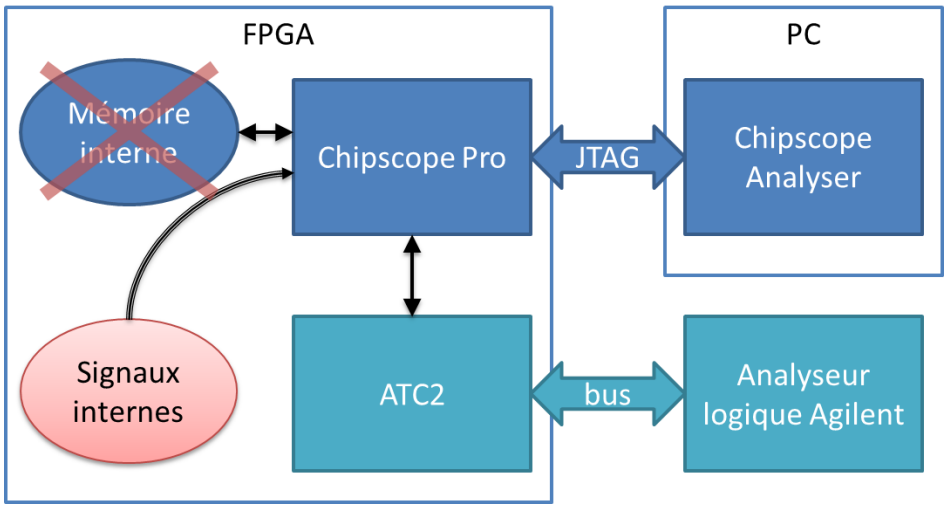


Figure 3.4 : Schéma de fonctionnement du module Chipscope Pro associé à un analyseur logique Agilent

Cette solution est un moyen de faire un compromis entre une solution intégrée et un analyseur logique performant. Il a été vu qu'un FPGA permettait un prototypage et une vérification des systèmes électroniques, mais il peut dans certains cas être limité en surface disponible. Ainsi, une autre solution est présentée dans la partie suivante : les systèmes d'émulation matérielle.

3.3 Systèmes d'émulation matérielle

Les systèmes d'émulation matérielle, ou émulateurs matériels, sont de gigantesques boîtes de prototypages pour les systèmes électroniques. Elles répondent aux besoins de beaucoup d'entrées/sorties, de surface et d'observabilité (Figure 3.5).

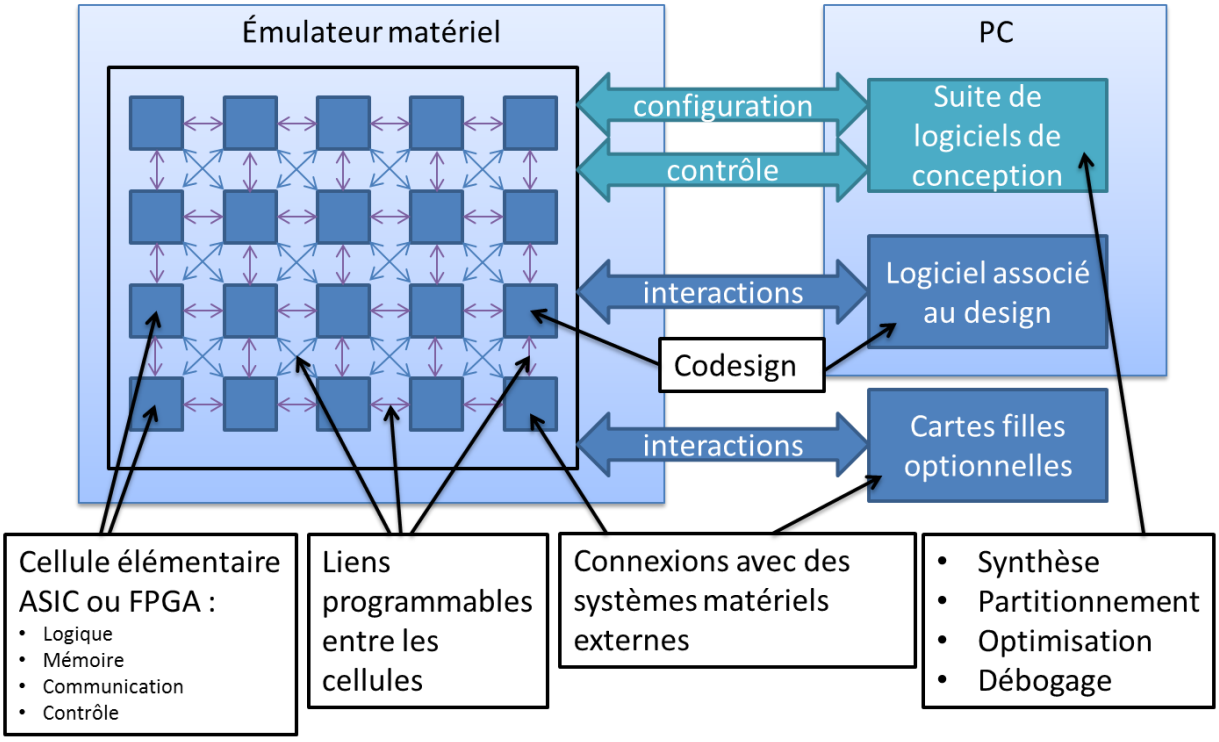


Figure 3.5 : Schéma de principe du fonctionnement d'un émulateur matériel.

Les intérêts de tels outils sont listés à la suite :

- **Vérification, intégration et débogage de blocs logiques** : à partir d'un ensemble de blocs logiques décrits en langage de description de matériel (HDL), le système désiré est représenté sur l'émulateur afin d'étudier son fonctionnement. Cette étape peut être réalisée bloc par bloc ou avec le système complet directement.
- **Covérification et intégration logicielle/matérielle** : en programmant l'émulateur tout en exécutant un logiciel sur un ordinateur connecté, la covérification intégrale du système peut être effectuée, prenant en compte le matériel et le logiciel. Les bancs de test supportent les langages standards C/C++, SystemVerilog et SystemC, mais aussi les langages spécifiques à la vérification par assertion : SVA [7] (System Verilog Assertions), OVL [16] (Open Verification Library), et PSL [11] (Property Specification Language).

- **Vérification et débogage de systèmes logiciels embarqués** : grâce à des liaisons JTAG ou des interfaces dédiées, le design de logiciels embarqués peut être vérifié sur des processeurs de cartes « filles » ou directement implémentés en langages de description de matériel (interfaces « in circuit emulator »).
- **Surface comparée à un FPGA** : pour les émulateurs matériels, la surface de travail en est de l'ordre de 100 millions à 1 milliard de portes logiques équivalentes tandis qu'elle n'est que de l'ordre de 1 million avec un seul FPGA. Les cellules d'accueil du système sont une matrice de FPGA ou d'ASIC (Application Specific Integrated Circuit). La Figure 3.6 donne un exemple d'une cellule ASIC qui est utilisée dans l'émulateur matériel Veloce de Mentor Graphics [17].

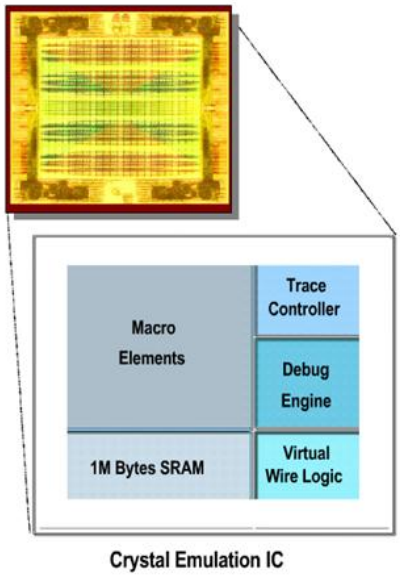


Figure 3.6 : Cellule ASIC (Application Specific Integrated Circuit) de l'émulateur Veloce de Mentor Graphics (image tirée de la présentation du produit Veloce par Mentor Graphics [18]).

- **Observabilité** : l'observabilité des signaux internes atteint 100 % du design sans avoir besoin de synthétiser à nouveau le système. Cette observabilité maximale nécessite cependant un ralentissement du système pour être opérationnelle.
- **Performances** : la fréquence d'émulation du système électronique émulé, avec observabilité complète, est de l'ordre de 5 MHz. La fréquence du système de l'émulateur est de l'ordre de celle d'un FPGA soit 500 MHz en moyenne.
- **Partage des ressources** : plusieurs émulateurs proposent des fonctionnalités de partage des ressources d'émulation entre plusieurs utilisateurs et plusieurs projets. Plusieurs systèmes peuvent donc être vérifiés simultanément.
- **Mémoire** : beaucoup de mémoire RAM est disponible pour le système, l'ordre de grandeur est de 100 GB. Elle permet de répondre aux besoins de stockage des données d'observation et aux besoins en mémoire des systèmes émulés sur l'outil.

Les systèmes d'émulation matérielle sont adaptés pour les projets de conception de circuits intégrés de grande envergure ou pour centraliser les ressources matérielles de plusieurs équipes. De plus, ces outils sont accompagnés de suites de logiciels qui permettent d'accélérer les cycles de production des concepteurs. Il y a notamment des outils de synthèse spécifiques, de partitionnement automatique et d'optimisation. Le flot de vérification d'un système électronique est semblable à celui qui est suivi avec un FPGA.

Les émulateurs matériels sont donc des solutions de prototypage et de vérification de haute performance. L'entreprise NVidia, célèbre concepteur de puces graphiques (GPU), a même présenté un article pour expliquer comment étaient prototypées ses cartes graphiques avec des émulateurs matériels [19]. Néanmoins, c'est une solution qui coûte cher, elle n'est pas à la portée de toutes les équipes de conception. Le prix moyen est de 0.02 $ [20] par porte équivalente soit 2 M$ pour une solution d'une taille de 100 millions de portes logiques équivalentes.

3.4 Méthodologie de vérification avec le WaferBoard™

Dans les moyens présentés pour effectuer la vérification de systèmes électroniques, la simulation logicielle et l'émulation matérielle, malgré leurs performances, sont confrontées à une limite essentielle : elles ne sont pas utilisables après la fabrication du système. De plus, ces solutions sont dépendantes des modèles matériels des circuits intégrés, il n'est pas possible d'avoir une vérification complète s'il manque le modèle matériel de certains circuits.

Pour pouvoir s'inscrire dans le contexte du WaferBoard™, dans lequel sont mis en jeu des circuits intégrés après fabrication, il est donc possible d'intégrer des circuits spécifiques à la vérification de systèmes électroniques. Ce sont des circuits d'accès aux signaux, pour leur contrôle ou leur observation. Cette méthodologie de conception est appelée DFT pour « Design for Test » ou « Design for Testability ». B. Vermeulen et al., discutant sur les tendances de la vérification des circuits intégrés, distinguent trois objectifs principaux de cette méthodologie DFT [21] :

- Une vérification de haute qualité : implique la caractérisation des performances des modules intégrés dans les circuits dédiés sur puce. Les designs à base de cellules de balayage sont les plus populaires, un des plus utilisés étant la norme JTAG, IEEE 1149 [22].

- Accessibilité : faculté à pouvoir observer les signaux internes d'un circuit à travers les entrées/sorties, sous la forme d'enveloppe de test (test *wrapper*) et de TAM (Test Access Mechanism). Les tests modulaires sont à privilégier, quand ils permettent d'isoler des circuits non logiques ou pour simplifier la démarche de la vérification (méthode itérative).
- Contrôle des signaux : faculté à générer des stimulis et des vecteurs de vérification, et faculté à évaluer les réponses de vérification. Il faut distinguer les générateurs de patrons sur puce des équipements de tests externes (ATE : Automatic Test Equipment). Le premier est adapté quand les patrons de vérification sont simples à générer, c'est-à-dire que la logique nécessaire n'occupe pas trop d'espace. L'avantage est d'avoir une plus grande vitesse, car il se trouve intégré aux circuits. Le deuxième est préféré quand la surface disponible est limitée ou que les patrons de vérification sont nombreux et complexes.

La Figure 3.7 illustre la méthodologie de vérification de systèmes électroniques avec le WaferBoard™. La spécification établie le point de départ pour permettre d'implémenter, d'intégrer et de fabriquer un système électronique. Un plan de vérification est dérivé dont l'objectif est de recenser l'ensemble des exigences de vérification, des tests à effectuer, des environnements de tests à développer pour s'assurer que le système fonctionne tel que spécifié. Quand un concepteur dispose des éléments qui constitue le système électronique, de l'environnement de vérification offert par le WaferBoard™ et du plan de vérification associé, il utilise le WaferBoard™ pour procéder à a vérification du système électronique à prototyper. Le WaferBoard™ doit disposer de circuits d'accès aux signaux (circuit DFT) pour pouvoir appliquer chaque test défini dans le plan de vérification. Pour chaque cas de test, des stimulis sont injecté pour contrôler le système et les signaux sont observés pour vérifier si le système réagit de la bonne manière. À chaque cas de test, le concepteur vérifie si le comportement est normal. Si oui, il peut alors passer au prochain cas de test. Sinon, il doit procéder au débogage de son système, c'est-à-dire qu'il doit identifier d'où provient l'erreur qui empêche le système d'avoir le comportement attendu. La vérification est terminée quand l'ensemble des cas de test définis par le plan de vérification se sont exécutés avec succès.

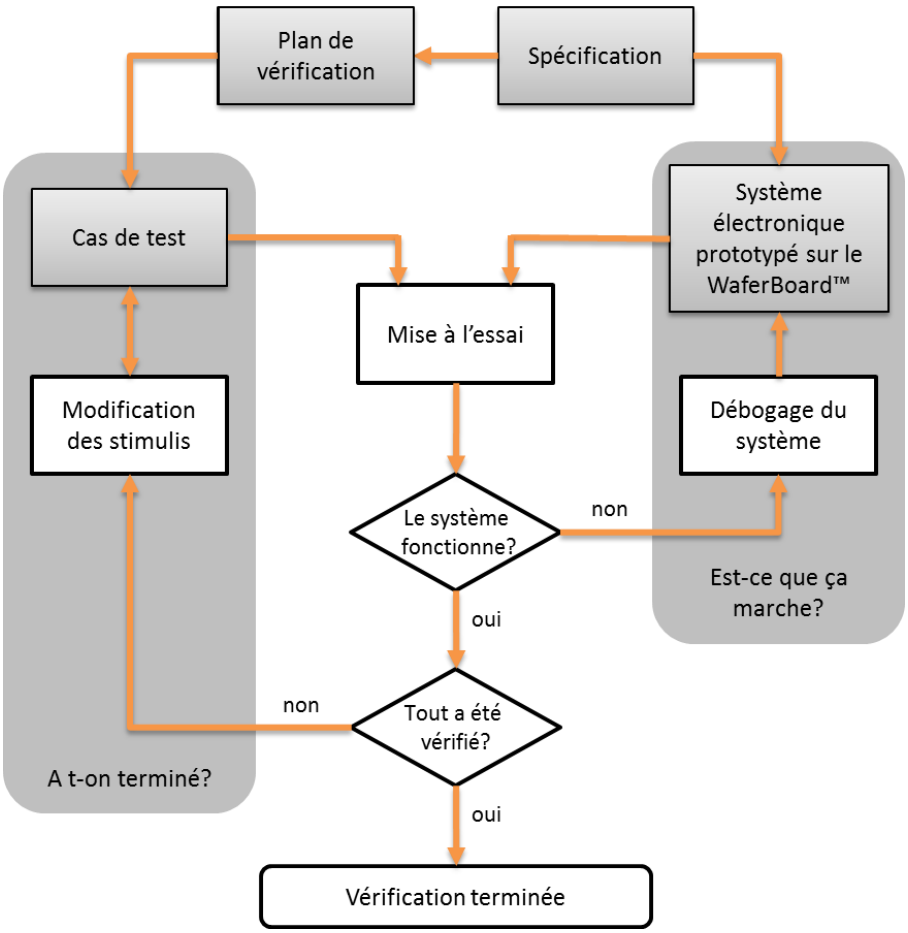


Figure 3.7 : Schéma de la méthodologie de vérification de systèmes électroniques avec le WaferBoard™.

3.5 Bilan sur les méthodes de vérification de systèmes électroniques

Cette partie résume les trois méthodes de vérification de systèmes électroniques qui ont été présentées dans la revue de littérature. Le Tableau 3.2 propose un résumé des principaux avantages et limites des différentes méthodes de vérification.

La simulation logicielle, l'implémentation sur FPGA ou l'implémentation sur émulateur matériel sont intéressantes pour effectuer la vérification d'un système électronique avant sa fabrication. Ces méthodes permettent ainsi de s'assurer qu'il n'y a pas d'erreur avant de fabriquer le système électronique en comparant les fonctionnalités obtenues avec les spécifications.

La simulation logicielle imite le comportement d'un système électronique sur un ordinateur à partir de modèles matériels des circuits intégrés. Elle a l'avantage d'offrir la meilleure

observabilité puisque le nombre de signaux qui peuvent être contrôlés ou observés est presque infini. Cependant, la totalité de la complexité du système électronique simulé doit être pris en charge par un processeur. Ceci entraine des temps de calculs longs qui ne permettent pas de vérifier un système électronique complexe de manière efficace.

L'implémentation sur FPGA ou sur un émulateur matériel permettent de créer un circuit physique réel dont les fonctionnalités sont identiques à celles du système électronique. Ces méthodes sont donc beaucoup plus rapides que la simulation matérielle. De plus, elles rendent possible l'identification des chemins critiques, c'est-à-dire les circuits logiques qui limitent les performances du système électronique. Les émulateurs matériels ont une surface et une observabilité beaucoup plus grande que celles des FPGA et peuvent être un meilleur choix dans le cas d'un système électronique complexe. De plus, ils disposent d'interfaces logicielles permettant de faciliter la création d'environnement de vérification pour la covérification (vérification mixte entre matériel et logiciel). Néanmoins le coût des émulateurs matériels est très élevé devant celui des FPGA, c'est un outil qui n'est pas à la portée de toutes les équipes de conception.

Tableau 3.2 : Avantages et limites des différentes méthodes de vérification de systèmes électroniques.

Méthode	Avantages	Limites
Simulation logicielle	• Avant fabrication • Observabilité	• Temps de calcul • Dépendance aux modèles • Pas de délais de propagation
FPGA	• Avant fabrication • Identification des chemins critiques	• Dépendance aux modèles • Surface limitée • Observabilité limitée • Pas de délais de propagation
Émulateur matériel	• Avant fabrication • Identification des chemins critiques • Observabilité • Surface • Covérification	• Dépendance aux modèles • Coût élevé : moyenne de 2 millions de dollars [3] • Pas de délais de propagation

Malgré les avantages que peuvent apporter ces trois méthodes de vérification de systèmes électroniques, elles ne peuvent pas modéliser les délais de propagation réels des signaux et sont aussi dépendantes à l'existence de modèles matériels des différents circuits intégrés. Enfin, pour pouvoir appliquer la méthodologie de vérification avec le WaferBoard™, ces différentes méthodes ne sont pas applicables. En effet, les systèmes électroniques mis en jeu avec le WaferBoard™ sont composés de circuits réels, fabriqué. Des circuits d'accès aux signaux doivent donc être considérés.

3.6 Circuits d'accès aux signaux

Pour que la méthodologie de vérification avec le WaferBoard™ soit applicable, il est nécessaire d'implémenter une solution adaptée pour l'accès aux signaux. Dans cette partie sera présentée la norme IEEE 1149.1 [22], JTAG, car elle est souvent retrouvée dans les circuits DFT. Puis un autre type de solution DFT sera décrit, basé sur des circuits logiques programmables. Enfin, il sera expliqué comment la norme IEEE 1500 répond aux besoins actuels dans le domaine des circuits DFT.

3.6.1 Norme IEEE 1149.1 : JTAG

La norme IEEE 1149.1, appelée JTAG, est une méthode de test des circuits normalisée en 1990 [22]. Elle fut instaurée pour tester les connexions et les soudures sur les PCBs, mais son architecture permet aussi de faciliter la vérification des fonctionnalités internes des circuits intégrés. Le principe est que les signaux d'entrées/sorties des différents circuits passent par des cellules dédiées JTAG enfouies dans les entrées et sorties des circuits intégrés. Ces cellules nommées BSCAN (Boundary Scan Cell) permettent d'observer et de contrôler les signaux qui y passent, ou simplement d'être transparente pour le circuit.

Un circuit intégré qui suit la norme JTAG contient un contrôleur TAP (Test Access Port Controller), qui est commandé par 5 signaux :

- TDI — *Test Data In* : entrée sérielle des données de test
- TDO — *Test Data Out* : sortie sérielle des données de test
- TMS — *Test Mode Select* : signal qui permet de choisir l'opération à effectuer sur les cellules JTAG

- TCK — *Test Clock* : horloge JTAG
- TRST — *Test Reset* : reset du circuit JTAG

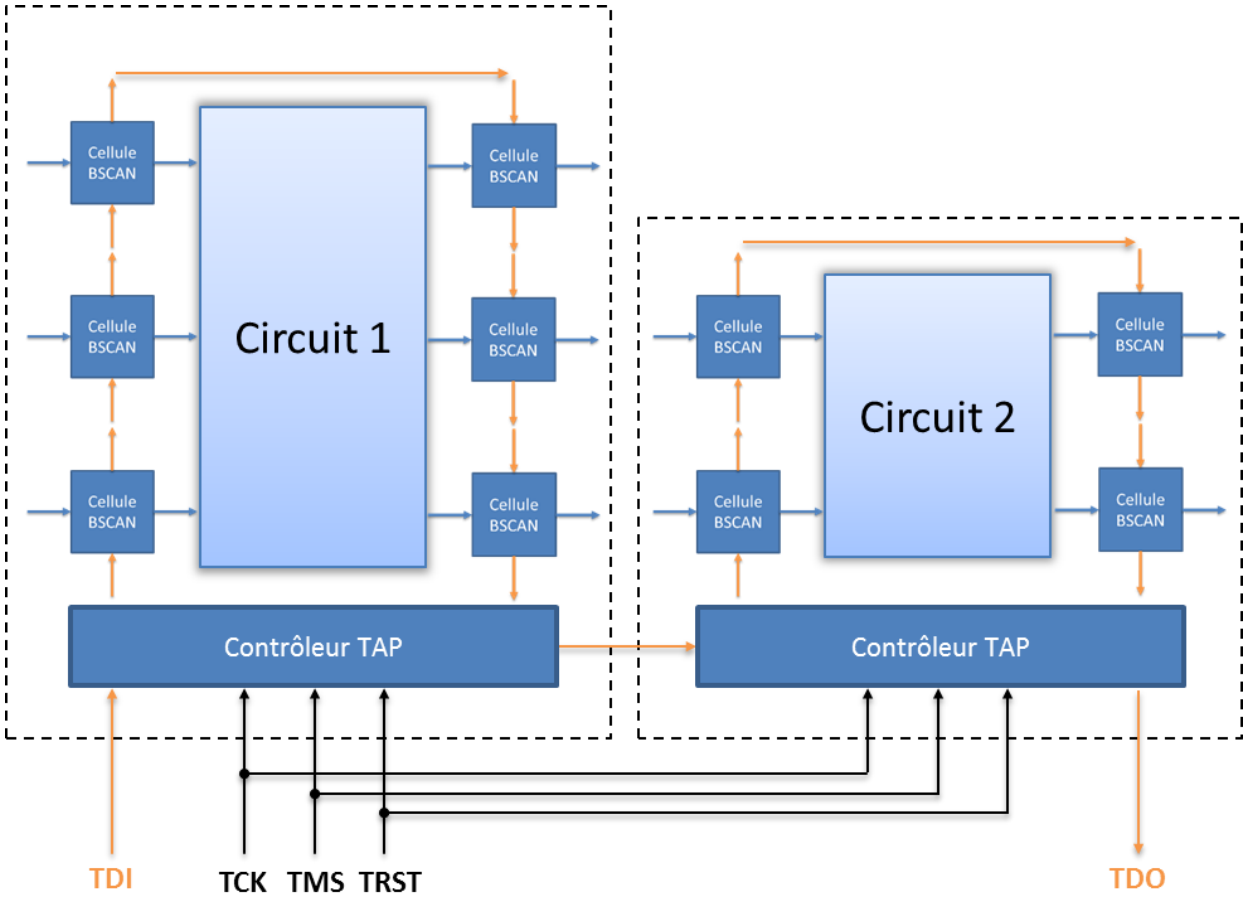


Figure 3.8: Exemple de circuiterie JTAG.

La Figure 3.8 donne un exemple de circuit JTAG. Chaque contrôleur TAP est relié aux signaux TMS, TCK et TRST. TDI n'est connecté qu'à la première cellule BSCAN, à travers le contrôleur TAP du premier circuit, puis se propage dans chaque cellule de chaque circuit du réseau JTAG, la sortie de la dernière cellule est connectée à TDO, à travers le contrôleur TAP du dernier circuit.

Le JTAG permet de se connecter à l'ensemble des entrées/sorties des circuits dans un système. Il est alors possible de générer des vecteurs de vérification aux sorties du circuit intégré et d'observer les entrées des autres pour vérifier les connexions et les interconnexions dans le PCB.

3.6.2 Circuits logiques programmables

Cette solution de Steve Wilton est d'utiliser des circuits logiques programmables (PLC pour « *programmable logic core* ») qui sont implémentés à l'intérieur des circuits intégrés [23]. Après la fabrication du système (*post-silicon*), il est possible d'observer des signaux internes du circuit intégré à travers le PLC. De plus, pour des objectifs de vérification plus complexes, ces derniers peuvent forcer des signaux internes pour observer les conséquences sur le fonctionnement. Cette idée se rapproche du principe déjà connu d'intégrer de la logique programmable dans des SOC (*System On Chip*) afin d'avoir une partie du design qui peut évoluer lors de la vérification après fabrication. C'est une approche évolutive de la conception d'un circuit intégré. Cependant, l'idée de Steve Wilton est de n'utiliser la logique programmable que pour des fins qui permettront le test du circuit intégré après fabrication.

Le principe de reprogrammabilité du circuit de vérification sert à pouvoir sélectionner la zone d'intérêt du circuit. En effet, les zones qui devront être réparées ne sont pas connues à l'avance. De plus, il est difficile d'obtenir un réseau fixe de vérification qui permet de voir n'importe lequel des signaux d'un design, étant donné la surcharge provoquée par les interconnexions supplémentaires. Ainsi, un réseau d'interconnexions et de concentrateurs est utilisé pour avoir accès à un grand nombre de signaux avec un impact minimum sur le routage. La communication des données avec l'extérieur doit être faite par un port JTAG ou via un processeur s'il y en a un de disponible sur le design.

L'impact de cette solution de vérification est modeste : en général inférieur à 5 % de la surface pour des ASIC de plus de 10 millions de portes. Le nombre maximum présenté de signaux internes accessibles à travers les concentrateurs est d'environ 7000. De plus, des fréquences de fonctionnement de plusieurs centaines de MHz ont été atteintes, permettant de s'approcher de la vitesse réelle des circuits vérifiés.

3.6.3 Plateformes de vérification IEEE 1500

La norme IEEE 1500 est dérivée du JTAG (IEEE 1149.1) et fonctionne sur le principe de cellules qui enveloppent les circuits ou sous-circuits à vérifier [24] (comme le JTAG). Elle a été créée pour les tests spécifiques aux circuits dans les systèmes sur puce (SOC, System On Chip). Il existe plusieurs travaux de recherche orientés sur la vérification qui utilisent la norme IEEE 1500

intégrée dans leurs circuits DFT. Par exemple, Kuen-Jong Lee et al., interfacent le bus d'un système ASIC avec un circuit IEEE 1500 pour démontrer les possibilités de vérifications modulaires sur les circuits sous test (vérification des sous-circuits un à un) [25]. Ils présentent des résultats positifs à la fois sur un FPGA de prototypage et sur une puce « réelle ». Autre exemple, Laung-Terng Wang et al., ont démontré la possibilité de réaliser la vérification d'un système complexe compatible avec la norme IEEE 1500 [26].

Puis en revenant à l'article de Bart Vermeulen et al., l'IEEE 1500 y est mis en valeur comme méthode d'accès aux circuits embarqués [21]. Quelques avantages par rapport à la norme JTAG y sont présentés :

- L'interface de données du JTAG (TDI/TDO) n'est que de 1 bit sériel tandis que l'IEEE 1500 permet une interface multibits parallèle.
- Les cellules BSCAN sont de 2 bits de large dans la norme alors que l'IEEE 1500 permet de choisir la taille des cellules à partir de 1 bit et plus.
- Le protocole JTAG est restreint au fonctionnement d'une machine à états fixes tandis que l'IEEE 1500 n'est pas limité à un protocole en particulier.

Ainsi, la norme IEEE 1500 a un protocole d'implémentation souple, ce qui permet qu'il soit mieux adapté à des situations de vérification spécifiques à un système électronique. De plus, l'utilisation de bus parallèles augmente la fréquence à laquelle la vérification peut fonctionner.

Enfin, la norme IEEE 1500 a un fort potentiel dans la création d'outils de vérification embarqués. Elle répond aux limites de fréquence de la vérification par la parallélisation et elle combine les avantages du protocole JTAG avec la possibilité de personnaliser des patrons de vérification ou des cellules. La section suivante est donc consacrée à une plus large revue de littérature sur cette norme.

3.7 Revue de la norme IEEE 1500

Développée de 1997 à 2005, la norme IEEE 1500 est une version étendue de la norme JTAG. Elle définit un procédé de test standardisé permettant de faciliter la vérification de systèmes sur puce (SOC, System On Chip) utilisant des modules réutilisés. Il répond aux besoins importants de la vérification sur puce qui sont la réutilisation et la performance. Dans cette partie, nous

proposons d'abord une revue technique de la norme P1500 qui permet de tracer les grandes lignes de l'architecture de la norme. Ensuite, nous étudierons les moyens de mise en œuvre à travers plusieurs articles. Enfin, nous présenterons les travaux et les résultats de quelques chercheurs trouvés dans la littérature récente. L'objectif de cette partie est de démontrer que la norme P1500 est adaptée pour créer une solution de vérification universelle, car elle est adaptable dans différents types de systèmes. Il est aussi intéressant de voir que plusieurs travaux s'attachent à optimiser l'utilisation de la norme pour la rendre plus efficace en tant que méthode de vérification de systèmes électroniques.

3.7.1 Revue technique

Fransisco Da Silva et al. [27] ont écrit un livre technique (handbook) sur la norme P1500. Il permet de traverser l'ensemble des détails de l'architecture et des protocoles. De plus, il propose un exemple de circuit IEEE 1500 qui est bâti au fur et à mesure de l'avancée du document. D'autres, comme Kim Petersén [28] ou encore Francisco Da Silva et al. [29], donnent des visions plus concises de la norme P1500 sous la forme d'articles.

Cette partie a pour vocation d'offrir un résumé technique pour permettre une meilleure compréhension des lectures présentées et des travaux réalisés. En effet, seuls les circuits et protocoles obligatoires de la norme sont présentés ici. Comme ils sont personnalisables pour des applications spécifiques il est important de connaître leur version d'origine. La solution de vérification pour le WaferBoard™ présentée dans ce mémoire est basée sur les concepts décrits dans cette section.

3.7.1.1 Introduction

La norme P1500 est constituée de trois modules principaux : le WIR (Wrapper Instruction Register), registre d'instruction, le WBR (Wrapper Boundary Register), registre d'enveloppe et le WBY (Wrapper Bypass Register), registre de *bypass*. La Figure 3.9 donne un schéma d'ensemble de l'architecture d'une enveloppe P1500. Le WIR est contrôlé par le bus d'entrée WSC et définit le fonctionnement du WBR et du WBY. Les données de test circulent entre l'entrée WSI et la sortie WSO (sériel) ou entre l'entrée WPI et la sortie WPO (parallèle).

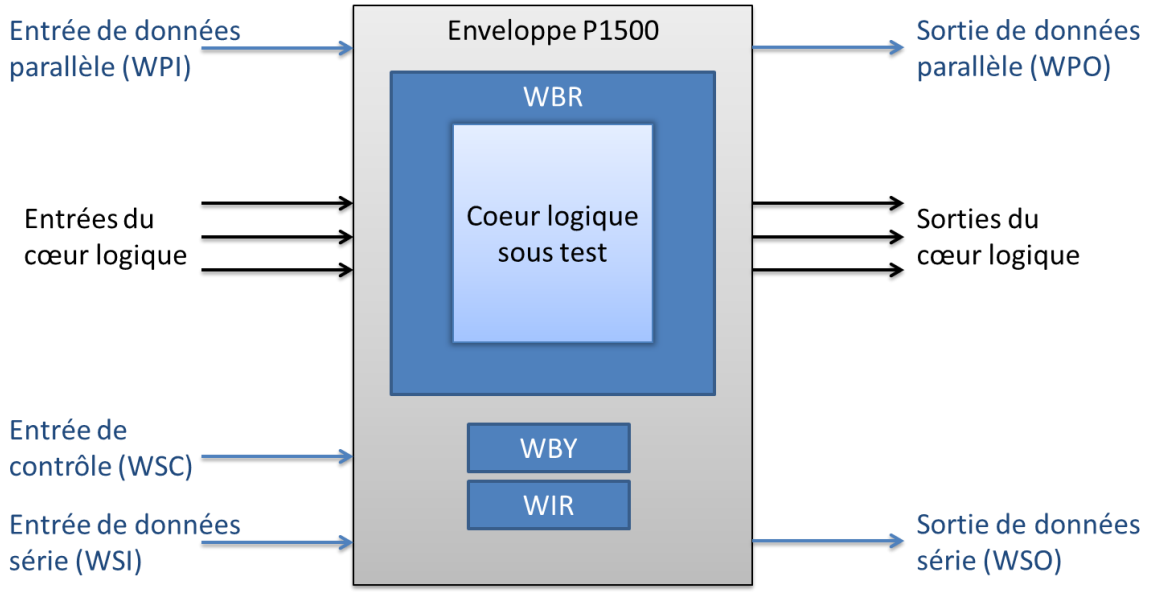


Figure 3.9 : Schéma de principe d'une enveloppe P1500.

Il existe trois modes de test obligatoires pour satisfaire la norme : mode de fonctionnement normal (BYPASS), de test interne (INTEST) et de test externe (EXTEST) :

- **Mode normal** : les cellules de l'enveloppe sont transparentes. Le P1500 n'a pas d'influence sur le fonctionnement des circuits logiques sous test. C'est le mode par défaut lors de l'initialisation du module.
- **Mode de test interne** (intest) : les entrées WSI ou WPI sont connectées à la circuiterie de test interne du cœur logique. Cette circuiterie peut être P1500 ou JTAG.
- **Mode de test externe** (extest) : les cellules forment une enveloppe autour du circuit logique, permettant un contrôle ou une observation des entrées/sorties. Ainsi, deux utilisations courantes de ce mode sont observées:
  - **Vers l'intérieur** (inward) : les cellules de l'enveloppe connectées aux entrées du circuit sous test sont contrôlées et les cellules connectées aux sorties sont observées. Ceci permet d'étudier le fonctionnement du circuit logique.
  - **Vers l'extérieur** (outward) : à l'inverse, les cellules connectées aux entrées sont observées et les cellules connectées aux sorties sont contrôlées. Ceci permet d'isoler le circuit logique du reste du système s'il y a lieu.

Ces modes correspondent à l'instruction active que l'on retrouve dans le registre d'instruction WIR. Dans les parties suivantes, on donne plus de détails sur les trois modules de l'enveloppe P1500.

3.7.1.2 WBR (*Wrapper Boundary Register*)

Le WBR est l'ensemble des cellules connectées aux entrées/sorties d'un circuit logique sous test, il forme une enveloppe (*wrapper*). Les cellules sont schématisées sur la Figure 3.10.

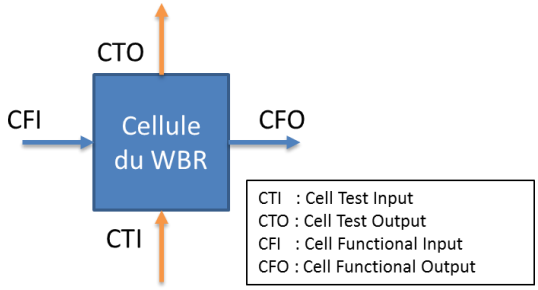


Figure 3.10 : Schéma d'une cellule du WBR d'un module P1500.

Il y a en général une cellule pour chaque entrée/sortie du circuit logique sous test. L'entrée/sortie du circuit logique traverse la cellule par le chemin fonctionnel (de CFI vers CFO).

Le chemin de test (de CTI vers CTO) est utilisé pour faire circuler les vecteurs ou résultats de vérification dans le système. Il y a deux configurations possibles pour le chemin de test :

- Configuration sérielle : l'entrée de données sérielle (WSI) est utilisée, les cellules sont connectées les unes à la suite des autres suivant le chemin de test, formant une chaîne unique (Figure 3.11).

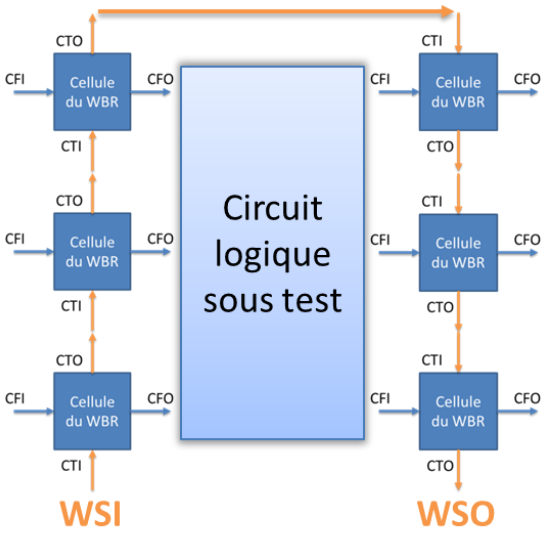


Figure 3.11 : Exemple d'un WBR utilisant un chemin de test série.

- Configuration parallèle : le bus d'entrée de données parallèle (WPI) est utilisé, plusieurs chaînes indépendantes de cellules existent en parallèle (Figure 3.12). Le nombre de bits de WPI est illimité.

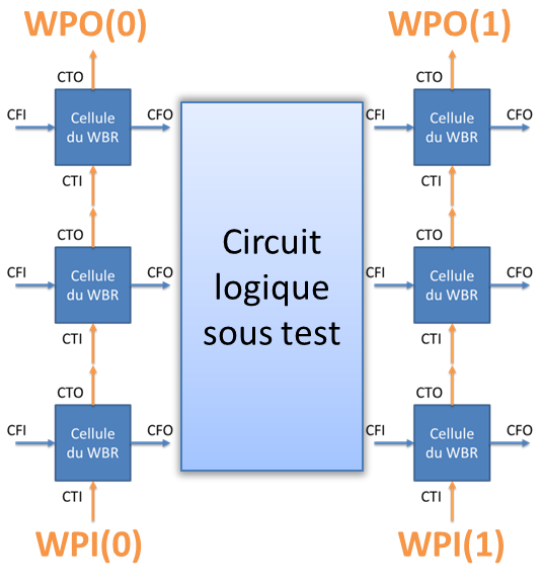


Figure 3.12 : Exemple d'un WBR utilisant un chemin de test parallèle.

La norme exige qu'au moins les trois opérations suivantes soient implémentées dans chaque cellule : fonctionnel, décalage (*shift*), capture (Figure 3.13).

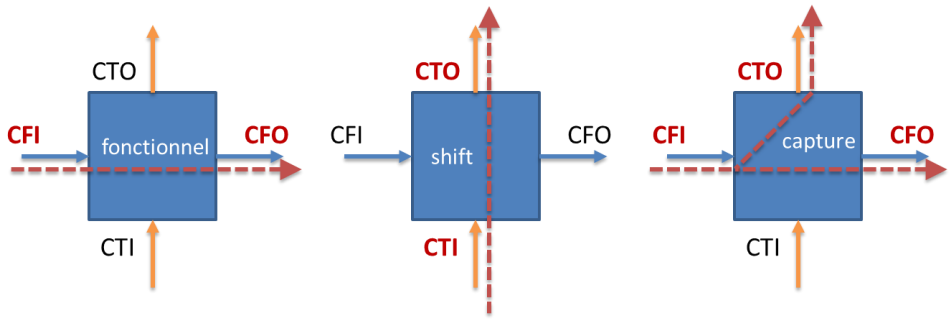


Figure 3.13 : Schéma des opérations standards dans les cellules du WBR — fonctionnel, décalage et capture.

- Fonctionnel : CFO=CFI. La cellule est transparente, les signaux traversent la cellule à travers le chemin fonctionnel de l'entrée CFI vers la sortie CFO;

- Décalage : CTO=CTI. Décalage du signal sur le chemin de test de l'entrée CTI vers la sortie CTO;
- Capture : CTO=CFI et CFO=CFI. Copie (observation) de la valeur de l'entrée fonctionnelle CFI vers la sortie de test CTO. Pendant ce temps, le chemin fonctionnel est préservé de CFI vers CFO.

Tant que ces opérations fonctionnent tel qu'attendu, les designs P1500 peuvent inclure autant d'opérations supplémentaires qu'il est possible d'en imaginer. Cependant, les opérations *update* (mise à jour) et *hold* (maintien), fréquemment rencontrées, sont présentées.

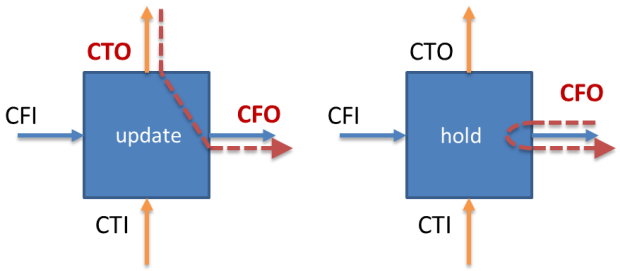


Figure 3.14 : Schéma d'opérations usuelles d'une cellule du WBR — update et hold

- *Update* : CFO=CTO. Copie de la valeur de la sortie de test CTO vers la sortie fonctionnelle CFO. Cette opération permet de contrôler des entrées fonctionnelles, pour envoyer des vecteurs de test par exemple.
- *Hold* : CFO=CFO. Blocage de la sortie fonctionnelle CFO sur sa valeur actuelle. Cette opération est notamment utilisée de manière implicite lors d'un décalage.

Finalement, pour sélectionner l'opération à effectuer dans les cellules ou encore pour choisir entre la configuration sérielle ou parallèle, des signaux de contrôle internes proviennent du registre d'instruction, le WIR.

3.7.1.3 WIR (*Wrapper Instruction Register*)

Le WIR est le décodeur d'instructions du P1500, il reçoit des instructions qui sont interprétées en signaux de contrôle internes vers le WBR et le WBY. Le WIR est représenté sur la Figure 3.15.

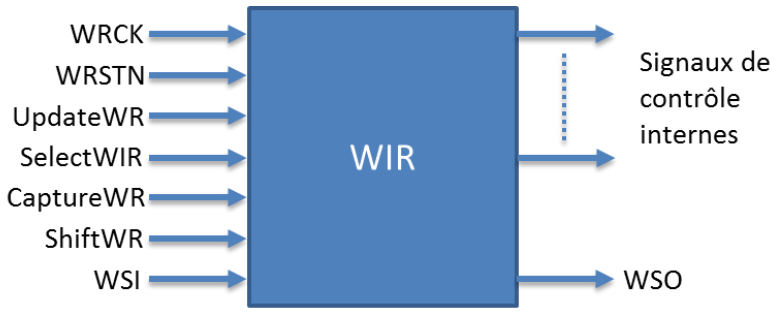


Figure 3.15 : Schéma du WIR d'une enveloppe P1500.

Le protocole à suivre pour charger un nouveau mode dans le WIR est précisément décrit dans la norme, avec un exemple à la Figure 3.16.

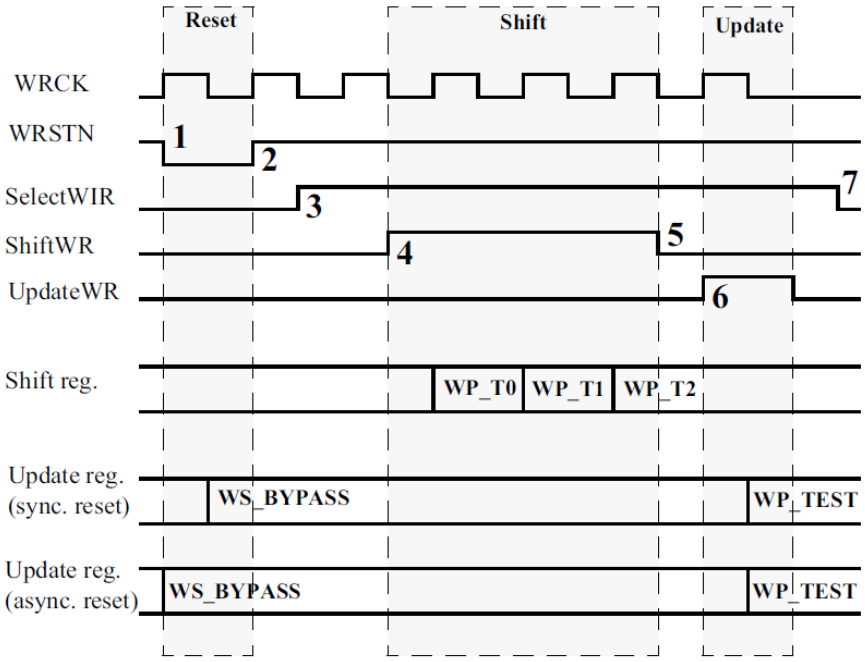


Figure 3.16 : Exemple de protocole de chargement d'une instruction de 3 bits (mode) dans le WIR (figure tirée de [27]).

Le signal d'entrée SelectWIR permet de choisir entre la configuration d'un nouveau mode ou le fonctionnement normal de l'enveloppe. Quand SelectWIR vaut '1', le WIR s'attend à recevoir une nouvelle instruction. Quand SelectWIR vaut '0', alors les différentes entrées du WIR permettent de sélectionner des opérations à effectuer sur les cellules. Ces opérations en fonction des entrées du WIR ont un effet qui peut varier en fonction du mode actif.

En général, il y a simplement :

- ShiftWR=1 : opération décalage
- UpdateWR=1 : opération update
- CaptureWR=1 : opération capture

3.7.1.4 WBY (*Wrapper BYpass*)

Dans un design avec P1500, il est fréquent d'avoir plusieurs circuits logiques ayant chacun leur WBR. Ils sont interconnectés de manière à ne former qu'un seul et même WBR qui enveloppe tous les circuits sous test. Cependant, les besoins de la vérification ne concernent pas nécessairement tous les circuits. Le WBY, registre de bypass, peut alors être utilisé pour réduire la longueur du WBR total (voir l'exemple Figure 3.17). Ce principe est le même que dans la norme JTAG.

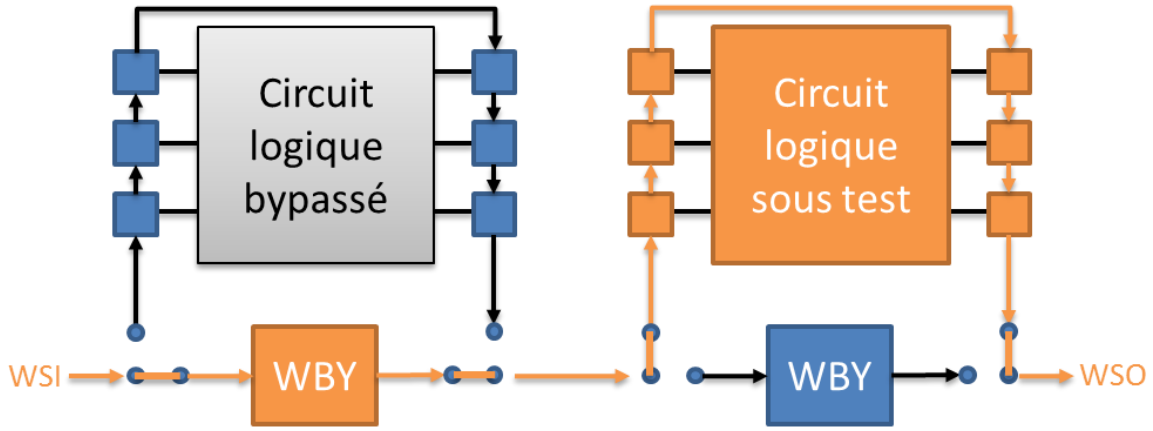


Figure 3.17 : Schéma d'exemple du principe du registre de bypass WBY, il n'y a ici qu'un seul circuit sous test.

Chaque circuit logique sous test possède un WBR et un WBY, mais il n'y a qu'un seul WIR pour tout le module P1500.

3.7.2 Mise en œuvre

Cette section donne un état de l'art sur la mise en œuvre et l'intérêt des circuits P1500.

3.7.2.1 Génération automatique

La génération automatique d'enveloppes IEEE 1500 est utile pour les concepteurs qui ne veulent pas avoir à gérer eux-mêmes la complexité de la norme. Elle permet aussi d'avoir plus de chance d'être conforme à la norme une fois le système généré et intégré au design. De plus, si des outils de génération automatique ont été développés, c'est que la technologie P1500 était un attrait pour les concepteurs de systèmes électroniques.

Krishna Chakravadhanula et Vivek Chickermane, de l'entreprise Cadence Design Systems, décrivent comment fonctionne leur outil de génération automatique P1500 pour le logiciel Encounter DTF Architect [30]. À partir d'une liste de description du circuit intégré à vérifier, une enveloppe P1500 spécifique est générée. Les configurations sérielles et parallèles sont disponibles ainsi que les modes INTEST (test interne du circuit) et EXTEST (test de la logique intercircuit). L'outil s'assure du respect de la norme ainsi que de l'intégrité des chaînes de registre de l'enveloppe grâce à un DRC (*Design Rules Check*). Enfin, il est possible d'assurer une compatibilité avec les outils DFT de compression des données. De la logique dédiée à la concaténation de chaînes est insérée dans l'enveloppe. La compression de données limite et homogénéise le volume de données pour chacun des circuits sous test. Les résultats présentés (enveloppe sérielle, parallèle puis à compression de données) démontrent une couverture de vérification très bonne (>=97 %) pour un surcoût en surface relatif à la surface et au nombre d'entrées/sorties du circuit sous test. Ce surcoût est estimé à environ 1 % et 23 % pour deux cas extrêmes.

Dans la même lignée, Wen-cheng Huang et al. présentent une plateforme de test automatique appelée DASTEP (*Design Automation System for SOC Test Platform*) compatible avec l'IEEE 1500 [31]. Les opérations de test des systèmes sont gérées par un processeur embarqué et un TAM (*Test Access Mechanism*) spécifique généré par l'outil. Un TAM est une méthode d'accès de l'information à un circuit, un protocole associé ou non avec un circuit. DASTEP permet aussi de rendre compatibles des circuits avec les normes JTAG ou IEEE 1500. De plus, un environnement et un flot de test sont créés et accessibles à travers une interface graphique dédiée. L'avantage de cette méthode est encore une fois d'avoir le moins de temps à passer sur la conception des infrastructures de tests. Néanmoins, cette méthode nécessite la présence d'un

processeur embarqué, d'un contrôleur de mémoire externe et d'une mémoire externe qui contient les données de test. Enfin, il n'y a pas de résultats concrets pour cette méthode.

La génération automatique d'enveloppe P1500 est mise à la disposition des concepteurs et se montre avantageuse pour réduire le temps de conception. Mais il est possible que certains systèmes nécessitent la création de solutions dédiées. Ainsi, plusieurs travaux seront présentés concernant la vérification de la compatibilité d'un design suivant la norme IEEE 1500. Ils pourraient être utilisés après la génération de l'outil de vérification pour le WaferBoard™ afin de s'assurer de leur conformité.

3.7.2.2 Compatibilité

La compatibilité à la norme est essentielle pour assurer que le fonctionnement de l'enveloppe IEEE 1500 sera conforme pendant l'exécution de la vérification d'un système électronique. Avant d'exposer des travaux qui s'attachent à ce problème, il convient de présenter un langage qui a été développé en parallèle avec la norme P1500. Il s'agit du CTL (*Core Test Language*), correspondant à la norme IEEE 1450.6 [32], qui permet de décrire les interfaces et caractéristiques d'une enveloppe attachée à un circuit logique. Rohit Kapur et al. ont écrit un article pour décrire en détail ce langage [33]. Ils montrent comment le CTL et l'IEEE 1500 sont étroitement reliés, quand un circuit logique décrit en CTL est vu comme une boîte noire et permet à un concepteur de créer une enveloppe compatible. Le CTL contient principalement trois types d'informations :

- Information sur les différentes configurations du design : décrit les différents modes de test dans lequel peuvent se trouver le circuit logique et son enveloppe. Il y a nécessairement les modes : fonctionnel (*bypass* de l'enveloppe), INTEST (test interne) et EXTEST (test de la logique externe).
- Information structurelle : décrit l'architecture des cellules de l'enveloppe et leurs fonctionnalités. Ces informations sont souvent réutilisables selon les besoins de vérification.
- Information sur les vecteurs de vérification : décrit les protocoles et les données associées à l'ensemble des vecteurs qui permettent la vérification du circuit logique.

Le CTL est finalement compatible quel que soit le type de circuit logique, la méthodologie de vérification ou la manière dont ce circuit est intégré dans le design.

A. Benso et al. proposent une solution pour vérifier la conformité à la norme P1500 des enveloppes et de leur description CTL associée [34]. Ils décrivent l'implémentation d'un environnement logiciel (*framework*) de vérification automatique qui permet aux vendeurs ou aux acheteurs de circuits logiques de vérifier le respect de la norme. Le principe de la vérification repose sur deux étapes, une vérification statique puis une vérification dynamique. La première consiste à valider la syntaxe de la description CTL et HDL de l'enveloppe avec un outil d'analyse de syntaxe qui a été développé à partir d'outils *open source*[7]. L'étude statique des descriptions permet aussi d'extraire et d'identifier la liste des caractéristiques de l'enveloppe étudiée. Cette liste est utilisée pour l'étape de vérification dynamique qui consiste à la vérification de l'enveloppe et du circuit logique par simulation. La simulation est prise en charge par un simulateur EDA (*Electronic Design Automation*) tandis que la vérification est assurée par le logiciel Specman Elite, spécialisé dans la comparaison entre des résultats de simulation et des spécifications.

Les travaux réalisés pour créer cet outil amènent à des résultats positifs dans le cadre d'un design simple : un compteur 4 bits. Il est montré que l'outil génère correctement la liste des caractéristiques et identifie chacune des erreurs que les concepteurs ont introduites volontairement.

Finalement, en plus de la norme IEEE 1500 qui permet de décrire des circuits dédiés à la vérification, un langage de description spécifique a été créé, le CTL. En outre, des méthodes de vérification de la conformité ont été pensées et validées. La norme P1500 est donc une base solide pour concevoir des circuits DFT et permet de réduire le temps de vérification.

3.7.3 Exemples d'applications

Cette section donne un état de l'art sur quelques applications du P1500 trouvées dans la littérature. Elle ne constitue pas une liste exhaustive, mais permet d'avoir un aperçu des différents champs d'orientation de recherche au sujet de la norme. D'abord, un article évaluant les

[7] JFLEX, un générateur d'analyseur de lexique de Vern Paxson, et CUP, un générateur de parser de Scott Hudson

avantages de la norme IEEE 1500 pour la vérification modulaire est présenté. Puis plusieurs travaux dont le but est d'améliorer les fonctionnalités des enveloppes P1500 pour des besoins spécifiques sont abordés.

3.7.3.1 Tests modulaires

Erik Marinissen et Yervant Jan Zorian expliquent que la norme P1500 n'est pas limitée à fournir une technologie permettant l'interopérabilité entre les systèmes [35]. Elle permet aussi de procéder à une vérification modulaire, c'est-à-dire à vérifier indépendamment les différents circuits d'un système. Cette méthodologie possède plusieurs avantages significatifs :

- Les circuits sur puce comportent de plus en plus de complexité à mesure que les procédés de gravure s'améliorent (la densité augmente). De plus, chaque circuit a des spécifications indépendantes en fonction de leur nature : processeur, FPGA, mémoire, modules de communication, etc. La norme P1500 facilite la construction des circuits DFT et réduit donc le temps de conception des interfaces de vérification.
- La vérification modulaire des circuits permet de réduire la quantité de données qui est générée, en moyenne de 50 % [36]. Ceci augmente donc la vitesse de la vérification.
- La vérification est dédiée à un circuit spécifique, mais si l'on doit vérifier un circuit de la même nature il est alors probable que les circuits DFT pourront être réutilisés partiellement. De plus, si un système contient plusieurs circuits identiques, les vecteurs de vérification n'auront à être générés qu'une seule fois.

3.7.3.2 Domaines d'horloge multiples

Yi Hyunbean et al. proposent une analyse de l'impact d'un système à plusieurs domaines d'horloge sur les circuits de vérification à base de cellule à balayage [37]. Plus précisément, ils démontrent qu'il est impossible d'arrêter et de redémarrer le système sans risquer d'invalider la cohérence des données dans le temps. Ils exposent deux types d'erreurs possibles. La première se produit quand le système doit être stoppé pour collecter les données de test, tous les circuits logiques ne seront pas bloqués en même temps, car il n'y a qu'une seule horloge de test, différente de l'horloge de certains circuits. Cette erreur est appelée invalidation des données (*Data Invalidation*). La deuxième est induite par le redémarrage des circuits après un arrêt. Pour les mêmes raisons, si les horloges sont différentes, il se peut que les différents circuits ne

redémarrent pas au même cycle. Cette erreur est nommée incompatibilité des phases d'horloges (*Clock Phase Relationship Inconsistency*).

La solution proposée est une cellule P1500 ayant trois modes de fonctionnement :

1. Vérification à simple domaine horloge : peut être utilisée quand le circuit logique n'est pas en relation avec un autre circuit ayant une horloge différente. Le circuit peut alors être arrêté et redémarré au cours de la vérification.
2. Vérification à multiple domaine d'horloge : utilisée quand le circuit sous test communique avec un ou plusieurs circuits de domaines d'horloge différents. Ce mode de fonctionnement utilise un détecteur d'invalidation des données et enregistre les états précédents pour pouvoir corriger les erreurs.
3. Vérification continue : permet d'observer les données sans arrêter les circuits sous test.

Les résultats montrent un impact moyen de 24 % en surface pour insérer de telles cellules sur différents circuits. Cependant, ces cellules spécialisées ne sont nécessaires qu'aux interfaces entre deux domaines d'horloge. Ainsi, l'impact pourrait être moins important à l'échelle d'un système, car de nombreuses interfaces n'ont qu'un domaine d'horloge.

Po-Lin Chen et al. discutent également à propos des problèmes associés à la vérification sur des circuits à multiples domaines d'horloge [38]. Ils proposent un ensemble de circuits de vérification, compatible avec des enveloppes P1500, qui permet de contrôler et d'observer des signaux à la fréquence du système. Un circuit spécifique gère la différence entre les horloges pour contrôler l'enveloppe P1500 et ainsi envoyer les commandes au bon moment et aux bonnes cellules. Ce circuit permet d'activer ou de désactiver des cellules données de l'enveloppe P1500 en fonction de si elles sont concernées par l'instruction courante. De plus, le circuit possède une fonctionnalité de mesure de délai qui avait été développée dans les travaux précédents des auteurs [39].

Les résultats démontrent que le coût en surface est comparable à d'autres solutions du même type (environ 500 cellules logiques), mais l'impact relatif au reste du design n'est pas connu. Un avantage spécifique à ces travaux est de proposer une solution qui s'adapte sans modifications à n'importe quelle enveloppe P1500. Un prototype a été validé avec succès en utilisant deux horloges à 50 et 25 MHz.

3.7.4 Bilan

On trouve de nombreuses autres applications de la norme IEEE 1500 dans la littérature. Par exemple, Yu-Jen Huang et Jin-Fu Li développent une méthodologie efficace et à faible surface pour vérifier l'interface entre un circuit logique et une mémoire de type RAM avec une enveloppe [40]. Enfin, Amitabh Das et al. proposent une enveloppe P1500 à accès sécurisé pour la vérification des systèmes dédiés à la cryptographie [41].

Ensuite, l'utilisation de la norme P1500 est en plein essor, car sa structure évolutive lui permet d'être amélioré selon les besoins de l'industrie et la recherche. Son objectif principal est de faciliter l'intégration des IP (Intellectual Property) dans les systèmes pour les vendeurs ou acheteurs de circuits. De plus, la norme P1500 facilite les tests modulaires qui réduisent le temps de vérification et permettent la réutilisation. Plusieurs outils existent pour aider à la mise en place de la norme dans un design et la littérature est riche en exemples.

Enfin, le P1500 est adapté pour créer une enveloppe pour la solution de vérification du WaferBoard™ décrite dans le prochain chapitre. Comme les systèmes électroniques qui seront implémentés sur le WaferBoard™ seront de nature variée, il est nécessaire que l'outil de vérification associé soit adaptable à chaque cas. Des outils permettraient également de vérifier la conformité des enveloppes P1500 générées pour une meilleure fiabilité. Enfin, grâce à la littérature disponible, l'utilisation de la norme favorise la compréhension de l'enveloppe de l'outil de vérification par l'utilisateur si ce dernier veut y intégrer des fonctionnalités spécifiques.

CHAPITRE 4 OUTIL DE VÉRIFICATION PROPOSÉ

Le projet de recherche décrit dans ce chapitre a pour objectif de proposer une méthode de vérification de systèmes électroniques compatible avec le WaferBoard™. Ce dernier permet de prototyper un système électronique en déposant les circuits intégrés qui le composent sur une surface dont les interconnexions sont reconfigurables. L'idée est donc d'ajouter un FPGA, contenant les fonctionnalités de vérification et de débogage, aux autres circuits intégrés du système. L'intérêt d'utiliser un FPGA est que la logique qu'il contient sera reconfigurable en fonction des situations de vérification rencontrées. L'outil de vérification de systèmes électronique, illustré par la Figure 4.1, doit être interfacé à un certain nombre de circuits intégrés en fonction des besoins de vérification. Il doit permettre l'observation ou le contrôle des signaux auxquels il est relié.

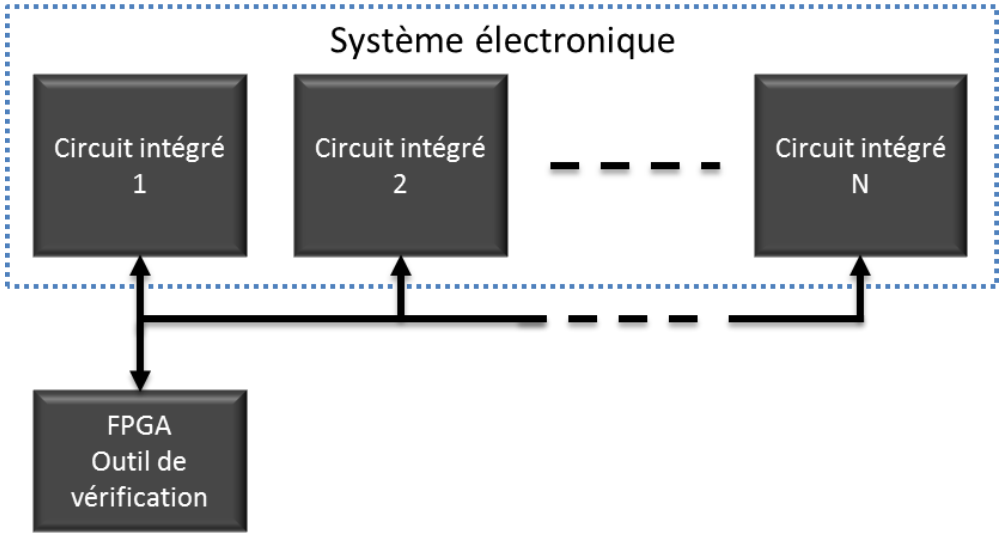


Figure 4.1 : Schéma de principe de l'outil de vérification.

Il est proposé de créer une enveloppe, contenant des cellules de contrôle ou d'observation de signaux, autour des circuits intégrés. Cette enveloppe sera construite à l'intérieur du FPGA à partir de la norme P1500. Comme il a été vu dans le Chapitre 3, Méthodes de Vérification, page 21, les avantages principaux d'utiliser le P1500 est de pouvoir adapter facilement les cellules de l'enveloppe en fonction des besoins de l'utilisateur et de disposer d'outils pour vérifier la conformité de l'enveloppe. De plus, la norme permet d'implémenter des solutions d'autres systèmes P1500 trouvés dans la littérature pour répondre à des besoins spécifiques.

Cependant, la possibilité de contrôler ou d'observer des signaux n'est pas suffisante pour former un outil de vérification. Il est aussi proposé d'inclure un analyseur logique dans l'outil qui permettra de configurer les différentes situations d'observation des signaux. Il sera associé à une mémoire interne permettant de stocker les informations prélevées lors du fonctionnement du système. Enfin, un module de gestion et de contrôle des différents modules de l'outil, répondant à des instructions de l'utilisateur, est nécessaire.

Quatre modules principaux constituent l'outil, ici interfacé à un seul circuit intégré d'un système électronique (Figure 4.2) :

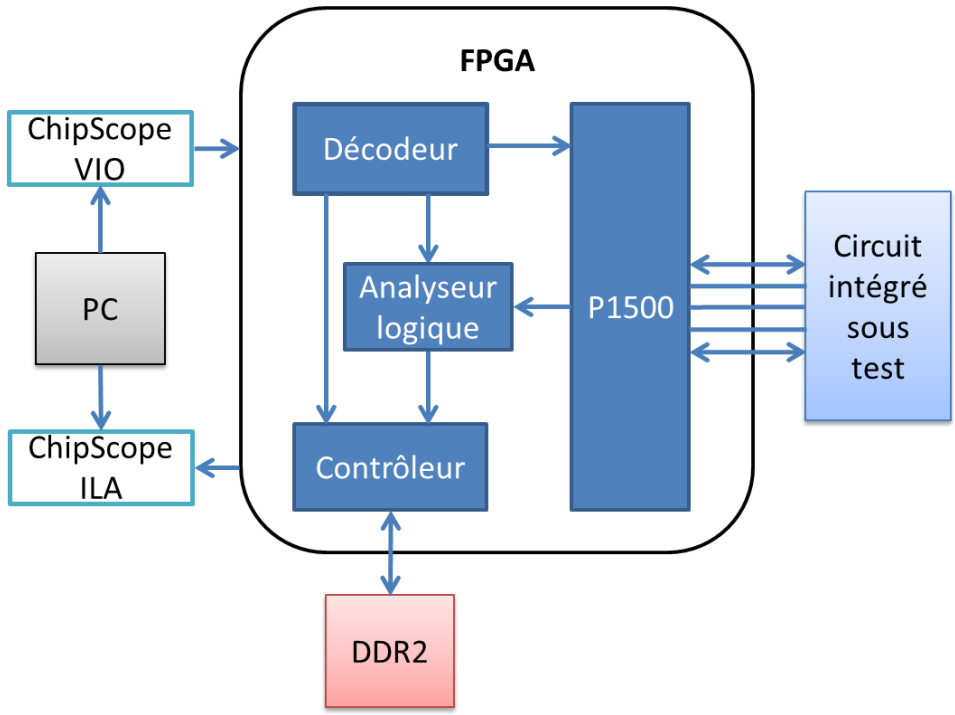


Figure 4.2 : Schéma de principe de l'architecture de l'outil de vérification.

- Décodeur : permet d'interpréter des commandes haut niveau pour le contrôle de la vérification. Les catégories de commandes sont :
  - Configuration des autres modules
  - Instructions P1500
  - Contrôle et acquisition de données
- P1500 : ce module constitue l'enveloppe physique dédiée à la vérification qui est créée autour des circuits logiques sous test. Il contient les modules :
  - Registre d'instruction
  - Cellules reconfigurables de contrôle et d'observation

- Analyseur logique : ce bloc permet de définir des conditions de déclenchement (logique) qui permettent de commencer ou d'arrêter l'échantillonnage de données en provenance du P1500.
- Contrôleur mémoire : ce module interfère avec une mémoire externe DDR2 pour le stockage des données.

Ce projet ne s'intéresse pas au mécanisme d'accès de test depuis un ordinateur. Les modules VIO et ILA de Xilinx ChipScope Pro sont utilisés pour s'interfacer avec l'outil de vérification et pouvoir le faire fonctionner. Le développement du système est proposé, en VHDL, sur une carte de développement ATLYS Spartan-6 avec mémoire DDR2 intégrée.

Ce chapitre présente d'abord les caractéristiques du projet et du design globaux. Ensuite, les spécifications et l'architecture de chacun des sous modules de l'outil vu sur la Figure 4.2 sont détaillées.

4.1 Fonctionnalités et généricité

4.1.1 Fonctionnalités

Le système réalisé possède un ensemble de fonctionnalités :

- Interactions avec un circuit logique sous test (Figure 4.3). Chaque entrée/sortie de chaque circuit traverse une cellule P1500. Cette fonctionnalité est utile pour vérifier l'ensemble du système.
    - Observation, sérielle ou parallèle, non intrusive des signaux sur les entrées/sorties
    - Contrôle, sériel ou parallèle, de la valeur de signaux

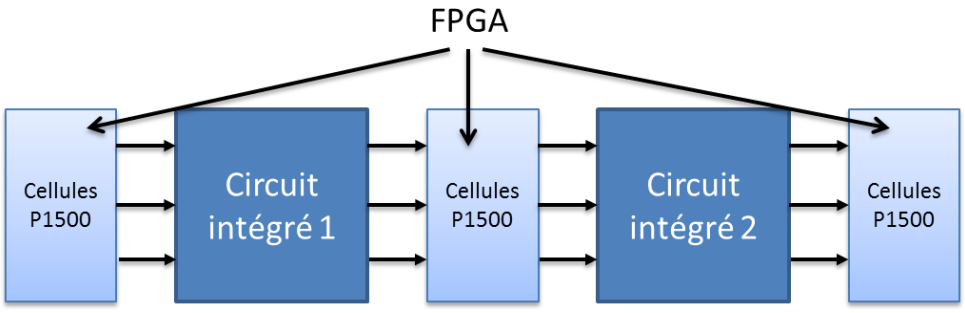


Figure 4.3 : Schéma d'exemple d'interactions entre des cellules P1500 et des circuits logiques sous test.

- Bypass d'un circuit logique sous test, permet de simuler le bon fonctionnement d'un design défectueux dans un système (Figure 4.4). Le circuit intégré 1 voit ses entrées/sorties traverser des cellules P1500 tandis que le circuit intégré 2 est isolé : ses sorties sont ignorées et remplacées par des signaux choisis par l'utilisateur. Cette fonctionnalité est utile pour une vérification modulaire du système.

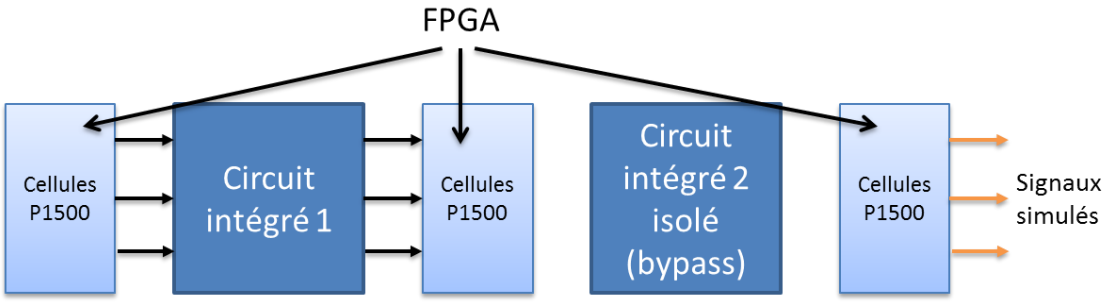


Figure 4.4 : Schéma d'exemple d'interactions entre des cellules P1500 et des circuits logiques sous test, le circuit logique de droite est isolé et ses signaux de sortie sont simulés par le P1500.

- Écriture et lecture de signaux dans une mémoire externe DDR2. Cette mémoire externe offre plus d'espace de stockage que la mémoire interne du FPGA. L'observabilité des circuits intégrés sous test est ainsi plus grande.
- Déclenchement de l'acquisition des données en fonctions des différentes conditions configurables de manière dynamique, à la manière d'un analyseur logique.
- Contrôle et configuration du système grâce à des instructions haut niveau.

4.1.2 Généricité

Le design présenté possède deux paramètres génériques (dans le fichier *debbuger_constants.vhd*) :

- Nombre de cellules de contrôle : `n_channel_con`
- Nombre de cellules d'observation : `n_channel_obs`

À partir de ces deux paramètres, tous les modules sont générés avec la configuration qui permet le fonctionnement immédiat du système. Cependant, le fichier de configuration des entrées/sorties doit être modifié en conséquence des choix précédents (fichier *debugger.ucf*). Ce fichier permet de relier les entrées/sorties du FPGA aux signaux logiques internes. Les correspondances changent donc en fonction du choix des interconnexions de l'outil avec les circuits intégrés sous test.

Les spécifications et l'architecture des sous-modules qui composent l'outil de vérification sont décrites à la suite.

4.2 Décodeur, spécifications et architecture

4.2.1 Spécifications du décodeur

Le décodeur est un bloc qui interprète des instructions haut-niveau provenant de l'utilisateur par Chipscope VIO afin d'obtenir la fonctionnalité voulue du système. Il transforme les instructions de haut niveau en signaux de contrôle compréhensibles par les différents sous-modules, le P1500, l'analyseur logique et le contrôleur. Le décodeur constitue l'interface du système avec l'utilisateur.

Les instructions codées sur les signaux **`opcode`** et **`data`** sont exécutées quand le décodeur détecte un niveau haut du signal **`exec`**. Le tableau suivant donne le codage et la description des différentes instructions. La première lettre des instructions désigne le module ciblé, W pour le P1500 (W pour *wrapper*), T pour le trigger (partie configurable de l'analyseur logique) et C pour le contrôleur.

Les instructions du décodeur (Tableau 4.1) sont exécutées en 1 cycle, sauf WIR_SET_MODE (changement de mode du P1500) et C_READ (lecture des données qui ont été écrites en mémoire depuis la dernière lecture). Pendant l'exécution d'une instruction multicycle, le décodeur ne permet pas l'exécution d'autres instructions.

Tableau 4.1 : Les instructions du module décodeur

INSTRUCTION	DESCRIPTION	Temps d'éxec.	Opcode	data
WIR_SET_MODE	Change le mode du P1500 selon le champ data	6 cycles	0000	0 : WS_BYPASS 1 : WS_EXTEST 2 : WP_EXTEST
WR_S_SHIFT	Décalage sériel du P1500	1 cycle	0001	bit 0 : WSI le reste : x*
WR_P_SHIFT	Décalage parallèle du P1500 Le nombre de bits du bus WPI dépend du nombre de cellules (paramètre générique)	1 cycle	0010	bit 0 : WPI(0) bit 1 : WPI(1) bit 2 : WPI(2) etc.
WR_UPDATE	Update du P1500	1 cycle	0011	x*
WR_CAPTURE	Capture du P1500	1 cycle	0100	x*
T_ARM	Opère un front montant du signal arm (active le trigger)	1 cycle	0101	x*
T_DEC_MODE	Paramètre le mode de déclenchement	1 cycle	0110	Voir la section 4.4.2.2
T_ WINDOW	Paramètre la taille de la fenêtre	1 cycle	0111	Voir la section 4.4.2.2
C_READ	Lecture des données qui ont été écrites en mémoire depuis la lecture précédente	n cycles	1000	x*
NOP	Rien		Tous les autres opcode	x*

*x : peu importe

4.2.2 Architecture du décodeur

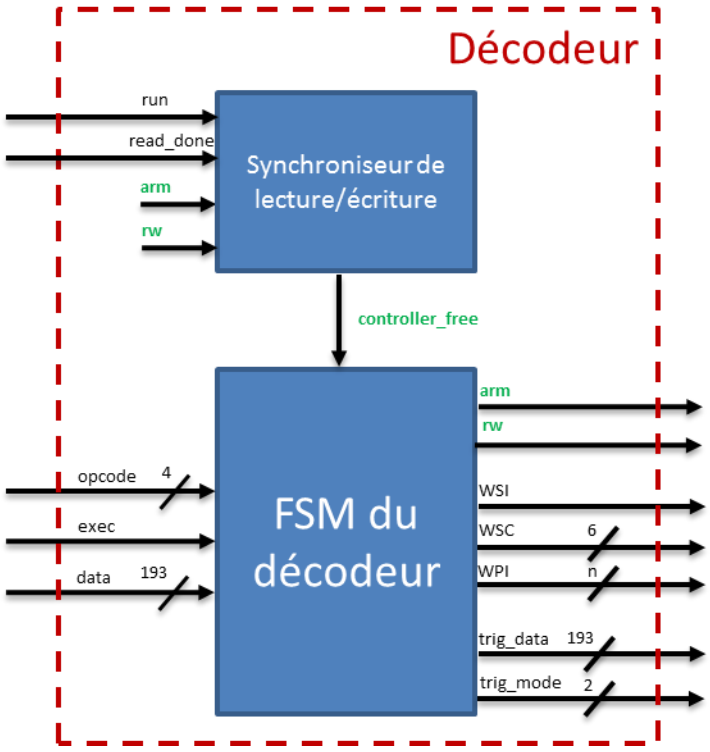


Figure 4.5 : Schéma bloc du module décodeur.

L'architecture du décodeur (Figure 4.5) est constituée de deux sous blocs, une machine à états (FSM) et un synchroniseur de lecture/écriture. La machine à états s'occupe d'affecter les signaux de sortie (à droite sur la figure) en fonction des instructions (signaux **`opcode`**, **`exec`** et **`data`**). Le synchroniseur de lecture/écriture a pour rôle de bloquer une demande de lecture (C_READ) si le système est en cours d'attente de déclenchement ou d'écriture. La lecture et écriture simultanée n'est pas implémentée dans ce système. Ainsi, l'instruction C_READ peut ne pas être prise en compte sous certaines conditions.

4.2.2.1 Machine à états finis du décodeur

La machine à états principale sélectionne d'abord le prochain état en fonction du signal **`opcode`** quand elle reçoit le signal **`exec`**, voir Figure 4.6 à Figure 4.10. Les groupes d'instructions de couleur différente sont représentés selon leur module de destination. Chaque instruction est un morceau de la même machine à état qui sera détaillée dans un diagramme.

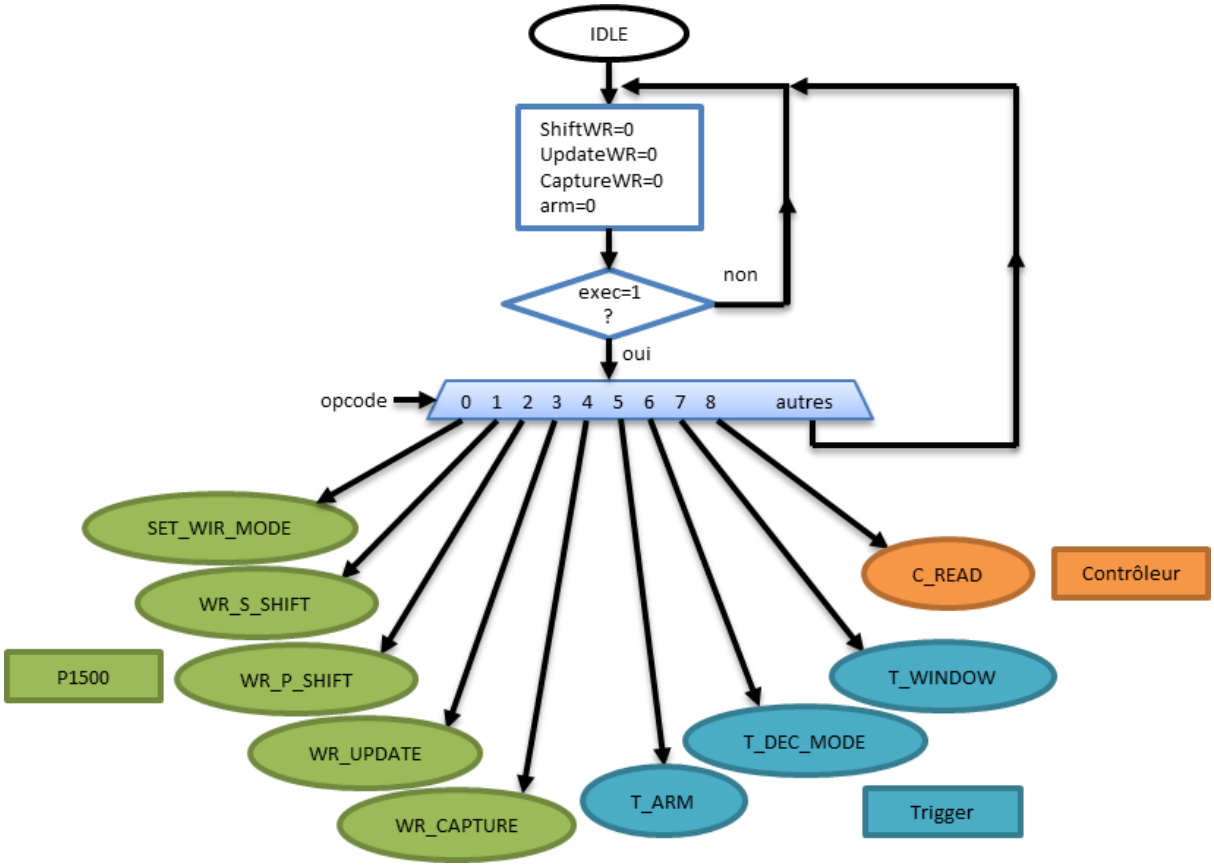


Figure 4.6 : Diagramme d'état de la machine à états finis du bloc décodeur (général) – 1er de 5.

L'instruction SET_WIR_MODE (Figure 4.8), qui permet de changer le mode du P1500, suit un protocole précis. Ce protocole est défini par la norme, voir la section 3.7.1.3, WIR (*Wrapper Instruction Register*), page 42.

L'instruction C_READ (Figure 4.10) est ignorée si le contrôleur est indisponible. Sinon, elle bloque le décodeur jusqu'à ce que toutes les données soient lues.

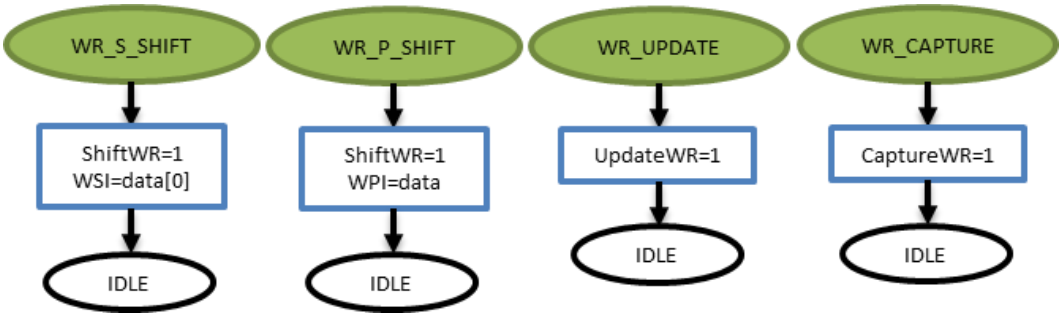


Figure 4.7 : Diagramme d'état de la machine à états finis du bloc décodeur (P1500) – 2ème sur 5.

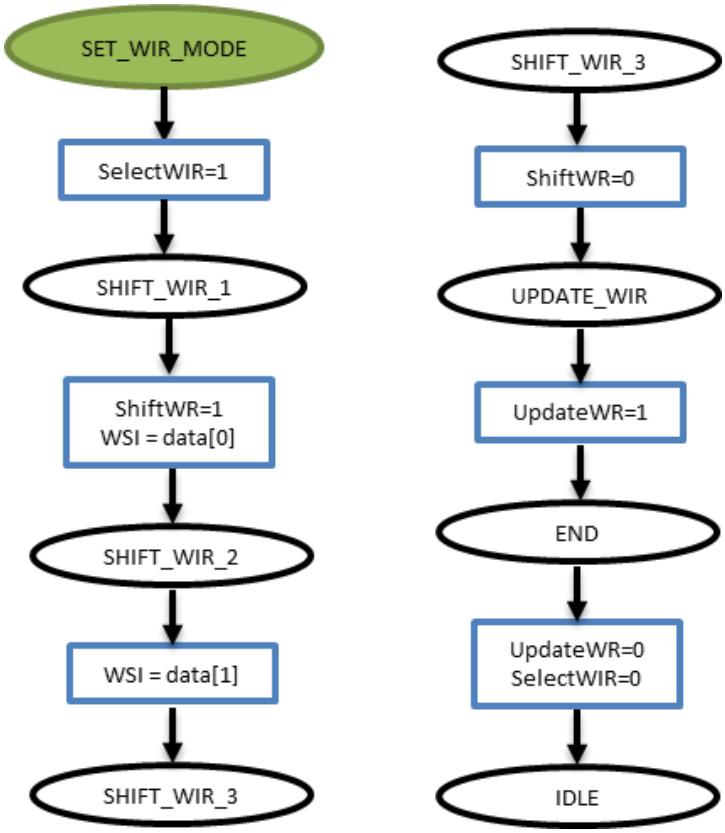


Figure 4.8 : Diagramme d'état de la machine à états finis du bloc décodeur (SET_WIR_MODE) – 3ème sur 5.

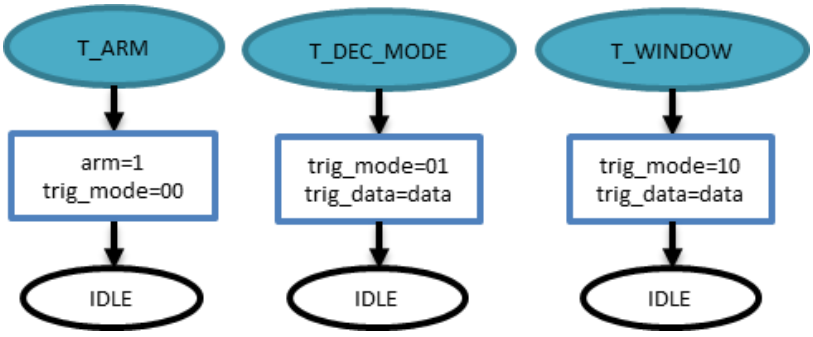


Figure 4.9 : Diagramme d'état de la machine à états finis du bloc décodeur (Trigger) – 4ème sur 5.

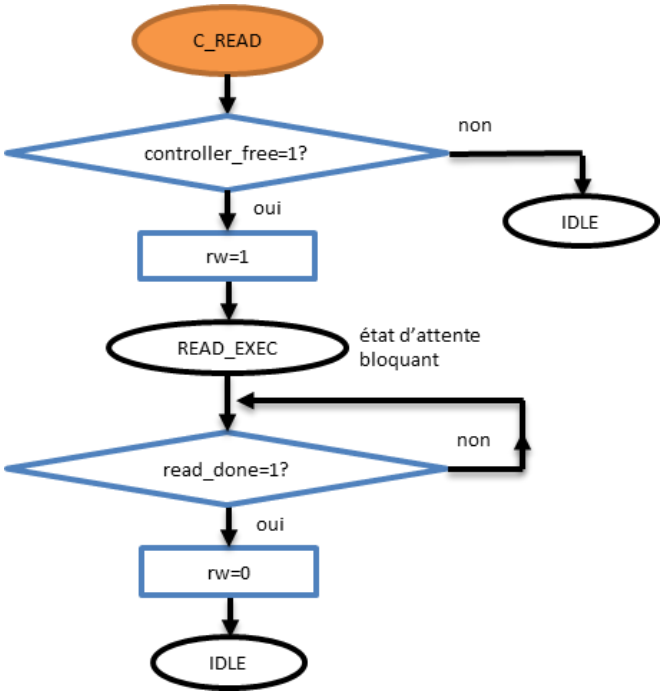


Figure 4.10 : Diagramme d'état de la machine à états finis du bloc décodeur (Contrôleur) – 5ème sur 5.

4.2.2.2 Synchroniseur de lecture/écriture

Le module synchroniseur de lecture/écriture est représenté par un diagramme d'états (Figure 4.11).

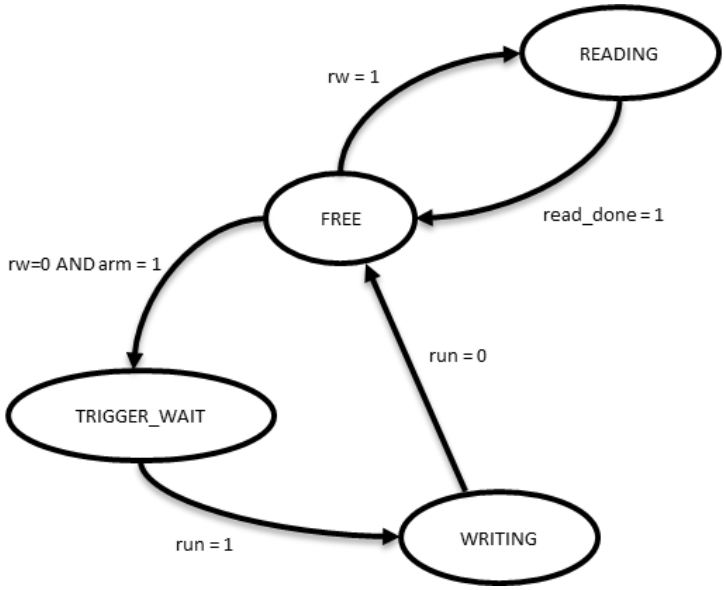


Figure 4.11 : Diagramme d'état du synchroniseur de lecture/écriture.

Ce module génère le signal **`controller_free`** qui indique à la machine à états finis du décodeur si l'on se trouve dans un état du système où le contrôleur mémoire est disponible, il a un rôle d'arbitre (Tableau 4.2).

Tableau 4.2 : Valeur du signal **`controller_free`** en fonction de l'état du synchroniseur.

ÉTAT	`Controller_free`
FREE	1
TRIGGER_WAIT	0
WRITING	0
READING	0

4.3 Enveloppe P1500, spécifications et architecture

4.3.1 Spécifications de l'enveloppe P1500

Le module P1500 a pour but de créer une interface de vérification performante. Il est basé sur la norme et respecte tous les aspects qui ont été décrits dans la section Revue technique, page 38.

4.3.1.1 Registre d'enveloppe WBR (*Wrapper Boundary Register*)

Le registre d'enveloppe WBR est constitué de deux types de cellules :

Les cellules d'observation : limitées à l'observation de signaux (Figure 4.12). Les opérations disponibles sont :

- Fonctionnel
- Décalage
- Capture

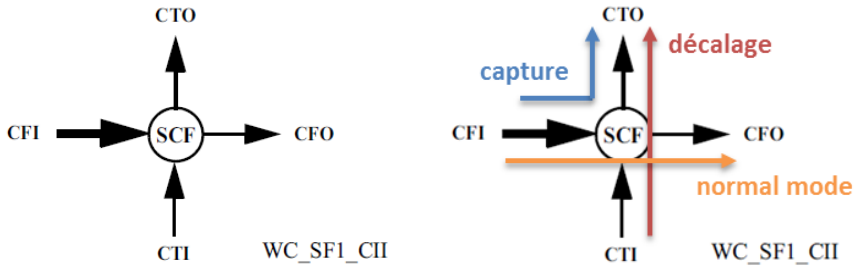


Figure 4.12 : Schéma d'une cellule d'observation du WBR — SCF : Décalage/Capture/Functionnal – tiré de [27].

Les cellules de contrôle : cellules permettant l'observation et le contrôle de signaux (Figure 4.13). Les opérations disponibles sont :

- Fonctionnel
- Décalage
- Capture
- Update

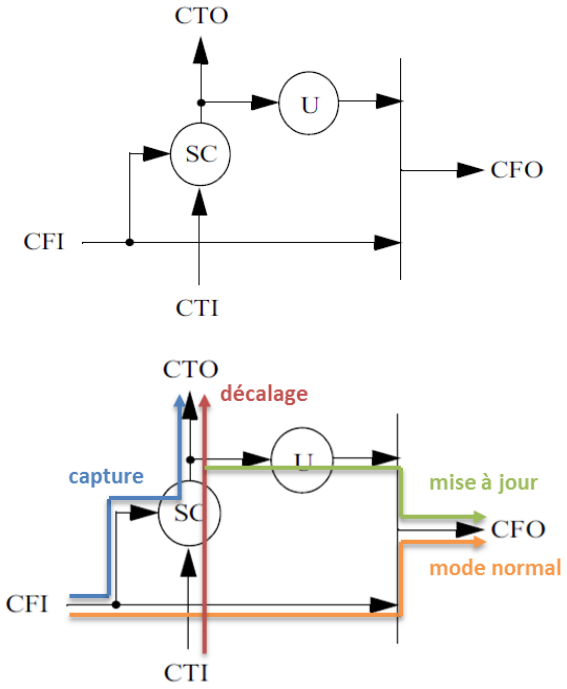


Figure 4.13 : Schéma d'une cellule de contrôle du WBR — SC : Décalage/Capture, U : Update – tiré de [27].

Pour chaque entrée/sortie reliée au module P1500, il est possible de choisir le type de cellule désiré.

Ce module implémente aussi un port d'accès parallèle pour le WBR pour maximiser les performances de vérification et se rapprocher des vitesses de fonctionnement des systèmes sous test. Le principe est d'avoir un bus de données de *n*-bits pour *n* cellules, formant ainsi *n* chaines de 1 bit. La Figure 4.14 est un exemple avec 6 cellules.

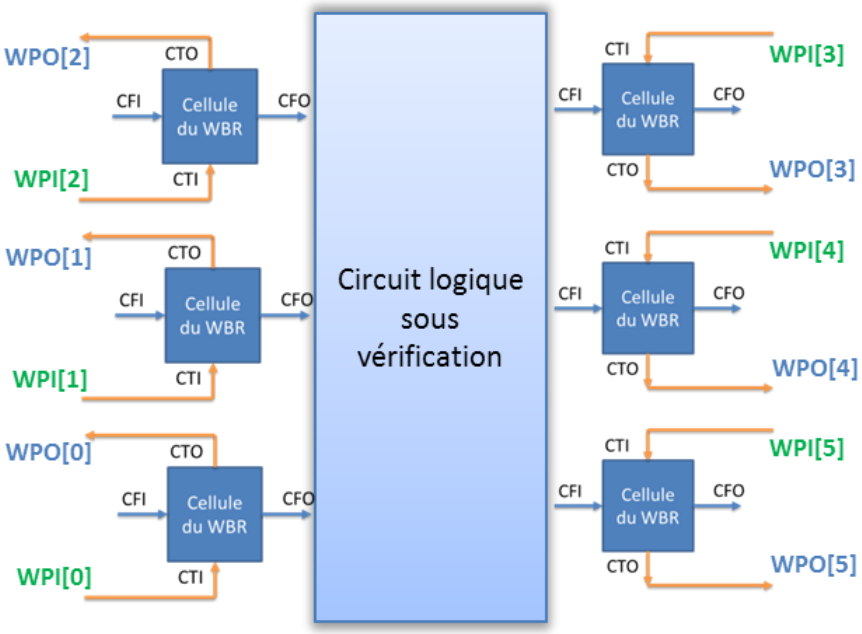


Figure 4.14 : Schéma d'exemple de l'accès de données parallèle pour un WBR à 6 cellules, avec WPI le bus d'entrée (6 bits) et WPO le bus de sortie (6 bits).

4.3.1.2 Registre d'instruction WIR (*Wrapper Instruction Register*)

Le registre d'instruction WIR dispose de 3 modes de fonctionnement :

- WS_BYPASS, mode par défaut, bypass, les cellules du WBR sont transparentes et la chaîne de test passe par le WBY. Ce mode devient actif lors d'un reset du système.
- WS_EXTEST, mode de test externe sériel, la chaîne de test passe par le port de données sériel du WBR
- WP_EXTEST, mode de test externe parallèle, la chaîne de test passe par le port de données parallèle du WBR

Les instructions du WIR sont codées sur 2 bits (3 instructions). Le codage des modes du WIR est présenté dans le Tableau 4.3.

Les opérations du WBR en fonction des signaux de contrôle et du mode du WIR sont données dans le Tableau 4.4. Ces informations permettent d'établir la logique nécessaire pour transformer les signaux de contrôle du WIR en signaux de contrôle internes pour le WBR et le WBY.

Tableau 4.3 : Codage des modes du WIR.

Code	MODE
00	WS_BYPASS
01	WS_EXTEST
10	WP_EXTEST
11	Non défini

Tableau 4.4 : Opérations du WBR en fonction des signaux de contrôle et du mode du WIR.

Signal de contrôle	**`ShiftWR` = 1**		**`CaptureWR` = 1**		**`UpdateWR` = 1**	
Cellule	contrôle	observation	contrôle	observation	contrôle	observation
WS_BYPASS	Fonctionnel	Fonctionnel	Fonctionnel	Fonctionnel	Fonctionnel	Fonctionnel
WS_EXTEST	Décalage sériel	Décalage sériel	Capture sérielle	Capture sérielle	Update	Hold
WP_EXTEST	Décalage parallèle	Décalage parallèle	Capture parallèle	Capture parallèle	Update	Hold

Les signaux de contrôle du WIR, `ShiftWR`, `UpdateWR` et `CaptureWR` ne peuvent pas être à l'état haut en même temps (une protection existe au niveau du WIR).

4.3.1.3 Registre de *bypass* WBY (*Wrapper BYpass*)

Le WBY est utilisé de la façon la plus simple en conformité avec la norme. Il devra permettre de bypasser ou non la chaîne du WBR en fonction du mode de fonctionnement.

4.3.2 Architecture de l'enveloppe P1500

4.3.2.1 Registre d'enveloppe WBR (*Wrapper Boundary Register*)

L'architecture des deux types de cellules du WBR est présentée sur la Figure 4.15 et la Figure 4.16. Ces cellules permettent respectivement d'effectuer les opérations définies dans la

spécification en fonction de signaux de contrôle, présentées dans le Tableau 4.5. À partir de ce tableau, il est possible de simplifier les signaux de contrôle :

- Pour les cellules des deux types, contrôle et observation, les signaux sont les mêmes en fonction d'une opération donnée
- On peut coder les signaux `shift_wr` et `hold_wr` sur un seul et même signal : `x_wbr[1]`
- On code `shift_en_wr` sur le signal : `x_wbr[0]`
- Le signal `update_wr` vaut désormais l'équation logique : `x_wbr[1]` AND $\overline{\texttt{x\_wbr[0]}}$

Pour le contrôle des cellules du WBR, un signal à 2 bits, x_wbr, est utilisé. Les valeurs du signal de contrôle `x_wbr` en fonction de l'opération voulue sont résumées dans le Tableau 4.6. La valeur des signaux de contrôle originaux en fonction de x_wbr est indiquée dans le Tableau 4.7.

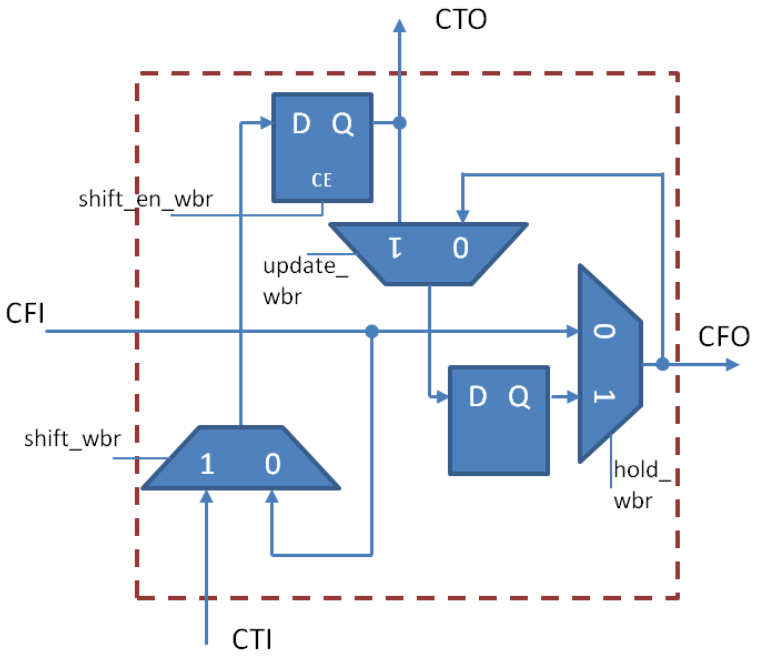


Figure 4.15 : Schéma de l'architecture d'une cellule de contrôle du WBR.

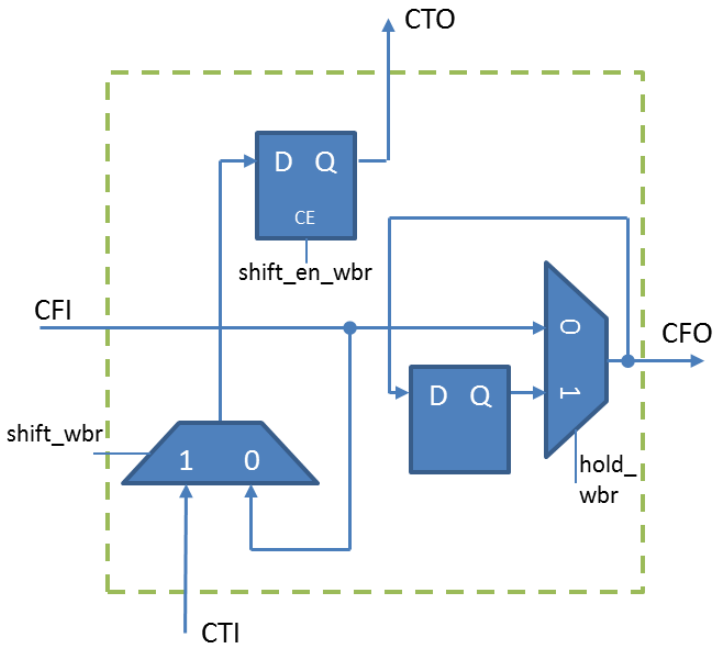


Figure 4.16 : Schéma de l'architecture d'une cellule d'observation du WBR.

Tableau 4.5 : Signaux de contrôle des cellules du WBR en fonction de l'opération voulue — x : peu importe, « - » : non concerné.

	Signal de contrôle	**shift_wr**		**hold_wr**		**update_wr**		**shift_en_wr**	
	Cellule	Contr.	Observ.	Contr.	Observ.	Contr.	Observ.	Contr.	Observ.
OPERATION	**Fonctionnel**	x	x	0	0	0	-	0	0
	Update	x	x	1	1	1	-	0	0
	Capture	0	0	0	0	0	-	1	1
	Décalage	1	1	1	1	0	-	1	1

Tableau 4.6 : Valeur du signal de contrôle simplifié des cellules du WBR en fonction de l'opération voulue.

	Signal de contrôle	x_wbr[1]	x_wbr[0]
OPERATION	**Fonctionnel**	0	0
	Update	1	0
	Capture	0	1
	Décalage	1	1

Tableau 4.7 : Valeur des signaux de contrôle originaux en fonction du signal x_wbr pour les cellules du WBR.

Signal de contrôle	shift_wr	hold_wr	update_wr	shift_en_wr
Valeur	x_wbr[1]	x_wbr[1]	x_wbr[1] AND $\overline{\text{x\_wbr[0]}}$	x_wbr[0]

4.3.2.2 Registre de *bypass* WBY (*Wrapper BYpass*)

Le registre de bypass, WBY, permet de relier WSO à WSI avec un seul registre (Figure 4.17). Son fonctionnement dépend du signal de contrôle shift_wby.

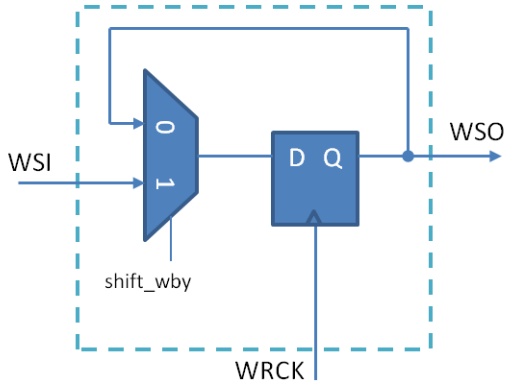


Figure 4.17 : Architecture du WBY.

4.3.2.3 Registre d'instruction WIR (*Wrapper Instruction Register*)

L'architecture du registre d'instruction WIR est présentée sur la Figure 4.18. Il y a les signaux de contrôle du WIR qui permettent de sélectionner les opérations des cellules en bleu sur le côté gauche du schéma. Le signal de données sérielles `WSI` permet de véhiculer les instructions (modes) du WIR à travers un registre à décalage de 2 bits. Les autres signaux sont des signaux de contrôles internes ou destinés au WBR et au WBY, ils sont décrits dans le Tableau 4.8.

Tableau 4.8 : Description des signaux du WIR.

Signal	Description
`shift_wir`	Active le décalage du registre à décalage, sert pour le chargement d'une nouvelle instruction dans le WIR. Fonctionne sur front descendant.
`Update_wir`	Met à jour le registre de l'instruction active du WIR à partir de la valeur présente dans le registre à décalage. Fonctionne sur front descendant.
`Mode_wir`	Registre de l'instruction active du WIR
`parallel_en`	Indique si le module P1500 doit fonctionner en mode parallèle. Actif à l'état haut.
`X_wbr`	Signal de contrôle des cellules du WBR.
`WSO_select`	Signal de sélection pour la sortie `WSO`, celle-ci peut provenir du WIR, du WBR ou du WBY.
`Shift_wby`	Signal d'activation du registre de bypass WBY. Actif à l'état haut.

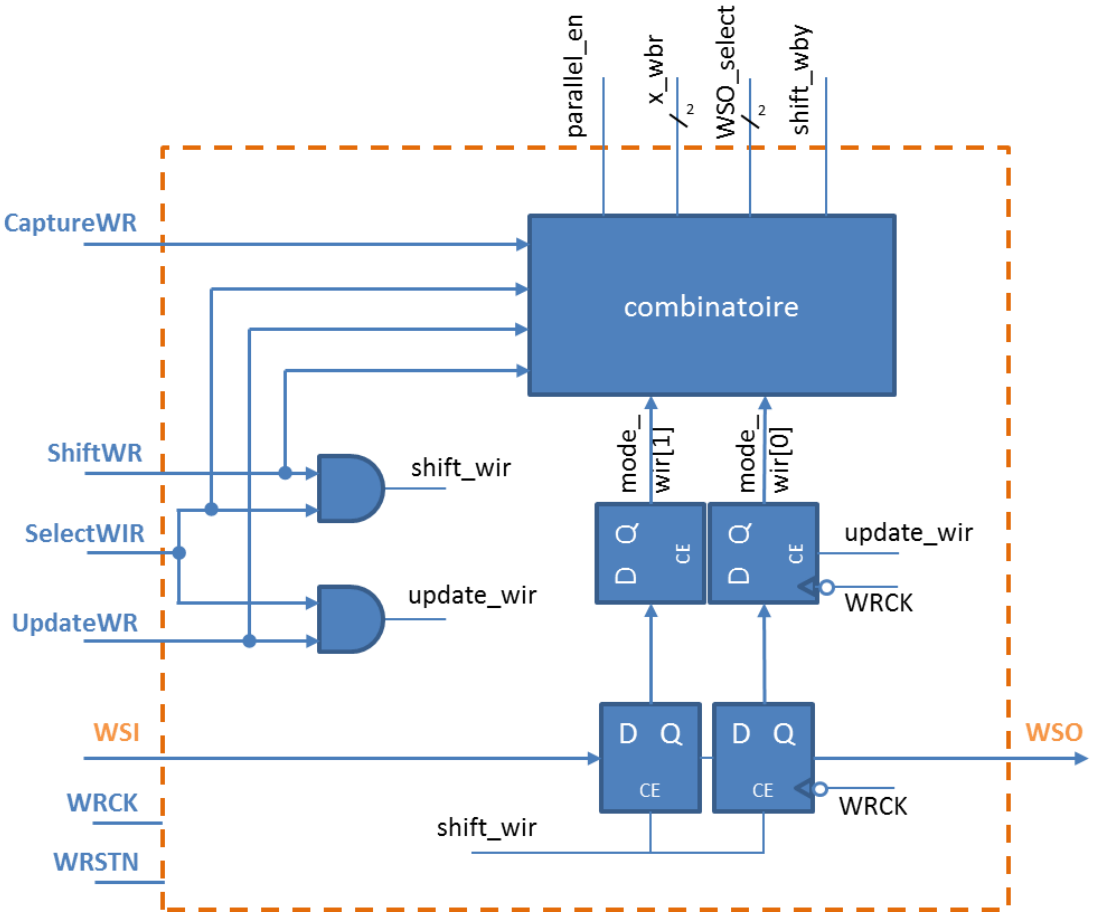


Figure 4.18 : Architecture du WIR.

Le Tableau 4.11 indique l'état de tous les signaux de contrôle selon les trois modes du WIR.

En général, le signal `WSO` provient du WIR lorsque celui-ci est sélectionné (signal `SelectWIR` à l'état haut), sinon, c'est le sous-module utilisé qui est choisi : WBY en mode WS_BYPASS et WP_EXTEST, WBR en mode WS_EXTEST (voir le Tableau 4.9).

Tableau 4.9 : Définition du signal `WSO_select`.

`WSO_select`	**Provenance de `WSO`**
00	WBR
01	WIR
10	WBY
11	WBY (par défaut)

Lorsque le WIR est en mode WS_BYPASS, l'entrée `WSI` est redirigée à travers le sous-module WBY et les cellules du WBR exécutent l'opération « fonctionnel ».

Lorsque le WIR est en mode WS_EXTEST ou WP_EXTEST, le signal `x_wbr` dépend des signaux `ShiftWR`, `UpdateWR` et `CaptureWR`. Lorsque plusieurs de ces signaux sont à 1, ce qui ne devrait pas arriver, la valeur qui correspond à l'opération « fonctionnel » est affectée. De plus, si le signal `SelectWIR` est à l'état haut, alors `x_wbr` doit encore correspondre à l'opération « fonctionnel ». La correspondance entre le signal `x_wbr` et les signaux d'entrée du WIR est présentée dans le Tableau 4.10.

Tableau 4.10 : Correspondance entre le signal `x_wbr` et les signaux d'entrée du WIR.

SelectWIR	**ShiftWR**	**UpdateWR**	**CaptureWR**	**x_wbr[1]**	**x_wbr[0]**	**Opération**
0	0	0	0	0	0	Fonctionnel
0	0	0	1	0	1	Capture
0	0	1	0	1	0	Update
0	1	0	0	1	1	Décalage
Autres cas				0	0	Fonctionnel

À partir de cette correspondance, une équation logique qui permet d'obtenir l'opération voulue est établie :

$$\texttt{x\_wbr[1]} = \overline{Select}\ AND\ \overline{C}(S \oplus U)$$

$$\texttt{x\_wbr[0]} = \overline{Select}\ AND\ \overline{U}(S \oplus C)$$

avec Select : `SelectWIR`

S : `ShiftWR`

U : `UpdateWR`

C : `CaptureWR`

Lorsque le WIR est en mode WP_EXTEST, le signal `parallel_en` est à l'état haut afin de diriger les signaux provenant du bus `WPI` vers les cellules du WBR.

Enfin, dans les modes WS_BYPASS et WP_EXTEST, le signal `shift_wby` est à l'état haut afin de faire passer les données sérielles (`WSI`) par le sous-module WBY.

Tableau 4.11 : Signaux générés par le WIR en fonction des signaux de contrôle en entrée.

	WS_BYPASS	**WS_EXTEST**	**WP_EXTEST**
parallel_en	0	0	1
x_wbr[1]	0	$\overline{SelectWIR}\left(\overline{C}(S \oplus U)\right)$	$\overline{SelectWIR}\left(\overline{C}(S \oplus U)\right)$
x_wbr[0]	0	$\overline{SelectWIR}\left(\overline{U}(S \oplus C)\right)$	$\overline{SelectWIR}\left(\overline{U}(S \oplus C)\right)$
shift_wby	1	0	1
WSO_select[1]	$\overline{\text{SelectWIR}}$	0	$\overline{\text{SelectWIR}}$
WSO_select[0]	SelectWIR	SelectWIR	SelectWIR

4.3.2.4 Architecture globale du module d'enveloppe P1500

L'architecture globale est représentée sur la Figure 4.19, c'est un exemple où l'on a quatre cellules P1500 de type indifférent. Il y a les trois modules principaux, le WBR, le WIR et le WBY. Les différents chemins de cette structure sont détaillés :

- Les chemins fonctionnels (vert) représentent les connexions directes avec l'extérieur, en général ce sont les liaisons avec les entrées/sorties de circuits logiques sous test.
- Le chemin de données sérielles (bleu clair) représente le chemin suivi par les données de vérification lors d'un test en série (de `WSI` vers `WSO`) comme dans le mode WS_EXTEST.
- Les chemins de données parallèles (rouge) représentent les chemins suivis par les données de vérification lors d'un test en parallèle (de `WPI` vers `WPO`) comme dans le mode WP_EXTEST.

- Les signaux de contrôle (bleu) représentent les signaux générés par le WIR pour contrôler l'ensemble des sous-modules. À ne pas confondre avec le bus `WSC` (noir) qui est l'entrée de contrôle du module P1500.

Enfin, un multiplexeur à la fin du chemin de données sériel permet de sélectionner le signal `WSI` qui ira vers `WSO`.

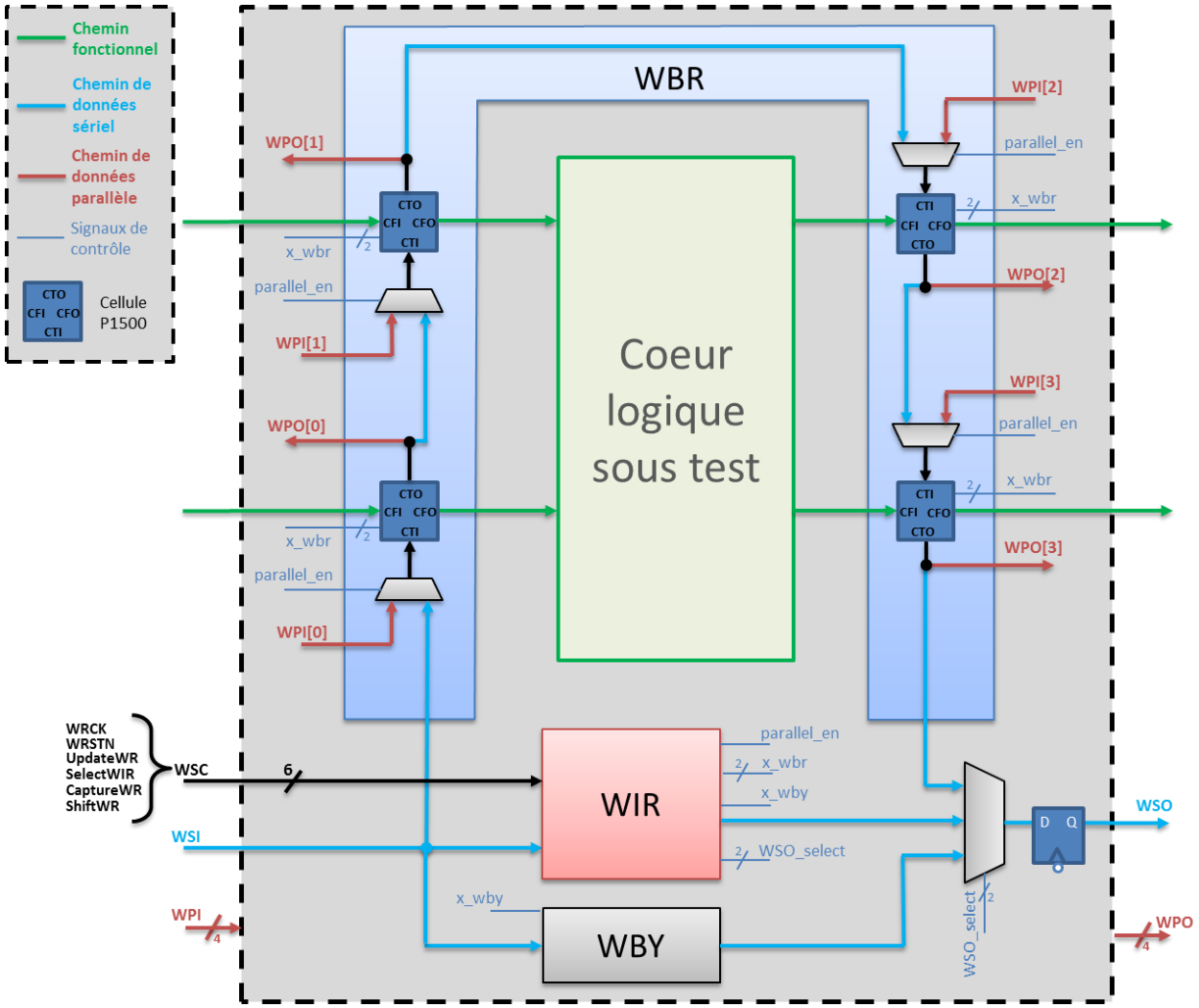


Figure 4.19 : Architecture globale du module P1500 — Exemple avec 4 cellules (type de cellule indifférent).

4.3.2.5 Généricité du module d'enveloppe P1500

Dans le système, le nombre de cellules de chaque type peut varier. Ceci a une influence sur la largeur des bus `WPI` et `WPO`. La Figure 4.20 représente la manière dont sont rangées les cellules sur les bus `WPI` et `WPO`. Les variables utilisées font référence à la section 4.1.2, Généricité, page 54.

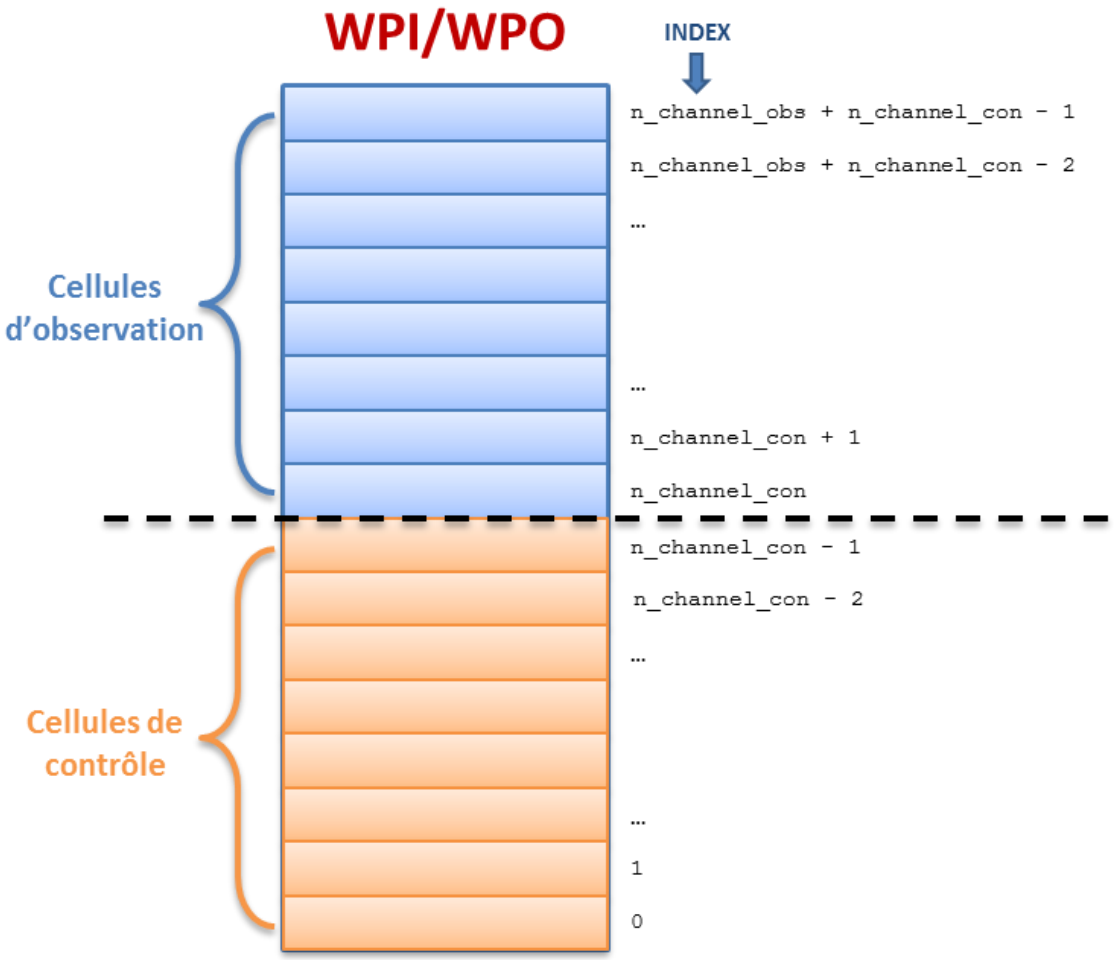


Figure 4.20 : Organisation générique des bus `WPI` et `WPO` en fonction du nombre de cellules.

Les cellules de contrôle se retrouvent dans la partie « LSB » du bus tandis que les cellules d'observation sont dans la partie « MSB ». Pour le signal sériel `WSI`, il est connecté à la première cellule de contrôle (équivalent à `WPI[0]`) tandis que `WSO` est connecté à la dernière cellule d'observation (équivalent à `WPO[n_channel_obs+n_channel_con-1]`).

4.4 Analyseur logique, spécifications et architecture

La structure de l'analyseur logique a été inspirée par le projet SUMP de Michael Poppitz [42], un design open source d'analyseur logique intégré sur FPGA.

4.4.1 Spécifications de l'analyseur logique

Ce module se place entre les cellules d'observation du WBR et la mémoire de stockage des données. Le but est d'étudier les signaux observés pour déclencher l'enregistrement des données en mémoire en fonction de conditions reconfigurables par l'utilisateur. L'analyseur logique réalisé dans ce système (Figure 4.21) offre les fonctionnalités de base suivantes :

- Échantillonnage à fréquence variable (*sampler*)
- Déclenchement d'acquisition en fonction de conditions variables et configurables (*trigger*)
- Gestion du temps d'acquisition (*trigger*)

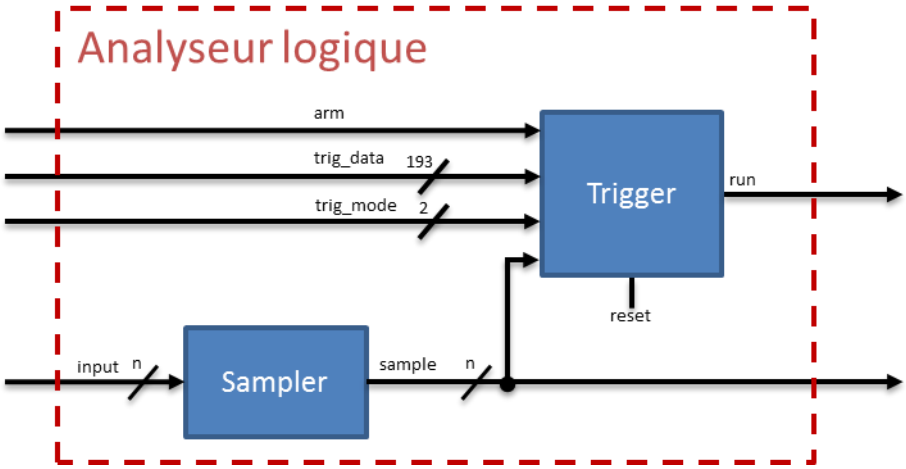


Figure 4.21 : Schéma de l'analyseur logique.

4.4.1.1 *Sampler*

Le module *sampler* est un échantillonneur à fréquence variable. Il permet de diviser la fréquence par une valeur comprise entre 2 et 16. Cependant, un échantillonnage à la fréquence du système a déjà lieu au niveau des cellules d'observation du WBR. C'est pourquoi le module *sampler* est simplement bypassé si l'on choisit de ne pas diviser la fréquence d'échantillonnage. La valeur de division n'est pas modifiable de manière dynamique, une reprogrammation du FGPA est nécessaire.

4.4.1.2 *Trigger*

Le *trigger* est un bloc de déclenchement configurable qui compare les échantillons provenant du sampler avec une ou plusieurs conditions de déclenchement. Quand il rencontre cette condition, il indique alors au bloc contrôleur qu'il est temps d'enregistrer les échantillons en mémoire (via le signal `run`). Le Tableau 4.12 donne les modes de configuration en fonction du signal d'entrée `trig_mode`.

Tableau 4.12 : Modes de fonctionnement du module *trigger* (analyseur logique).

`trig_mode`	**Mode de fonctionnement**
00	Fonctionnel
01	Configuration du déclencheur
10	Configuration de la fenêtre
11	Non défini

Dans les sous parties suivantes, les spécifications des différents modes du *trigger* sont présentées.

4.4.1.2.1 Configuration du déclencheur

Dans ce mode, les conditions de déclenchement sont spécifiées en fonction des canaux d'entrée via le signal `trig_data`. Cette valeur de configuration est enregistrée dans le registre de déclenchement de 193 bits (Figure 4.22).

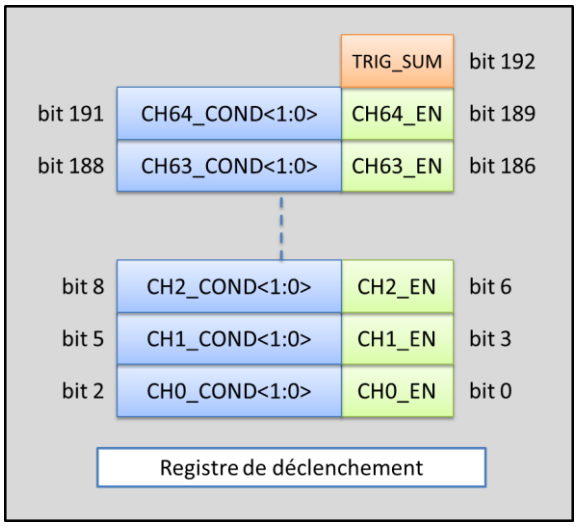


Figure 4.22 : Détails du registre de déclenchement.

On donne la signification de chaque bit de ce registre dans le Tableau 4.13.

Tableau 4.13 : Signification des bits du registre de déclenchement du trigger.

bit 192	**TRIG_SUM** : association des conditions de déclenchements des différents canaux d'entrée 0 = ET logique 1 = OU logique
bit <1 :2>+3i	**CHx_COND<1:0>** : condition de déclenchement du canal d'entrée x 00 = état haut 01 = état bas 10 = front montant 11 = front descendant
bit 0+3i	**CHx_EN** : activation du déclenchement selon le canal d'entrée x 0 = désactivé 1 = activé
i : index du canal	

En résumé, la condition de déclenchement peut associer une condition pour chaque canal (jusqu'à 64 maximum) avec la logique « ET » ou « OU ». Le déclencheur est limité à des conditions d'un cycle d'horloge (pas de conditions multicycles).

4.4.1.2.2 Configuration de la fenêtre

Dans ce mode est spécifié le temps d'échantillonnage qu'on désire à partir du déclenchement, via le signal `trig_data` (le nombre de cycles d'horloge pendant lequel se déroule l'enregistrement). Cette valeur de configuration est enregistrée dans le registre WINDOW de 24 bits (Tableau 4.14).

Tableau 4.14 : Signification des bits du registre WINDOW (trigger).

bits 23-0	**WINDOW<23:0>** : temps d'échantillonnage en cycles d'horloge

4.4.1.2.3 Mode fonctionnel

Dans le mode fonctionnel, le module *trigger* attend le signal de mise à feu (via le signal `arm`) pour se mettre en observation de la condition de déclenchement. Ensuite, quand la condition est rencontrée, le module spécifie au contrôleur mémoire d'enregistrer les données pendant la durée correspondant à la valeur du registre WINDOW (via le signal `run`).

4.4.2 Architecture de l'analyseur logique

4.4.2.1 *Sampler*

L'architecture du *sampler* (Figure 4.23) est faite pour fonctionner dans 2 modes différents :

- Mode diviseur : le *sampler* joue le rôle d'un diviseur de fréquence d'échantillonnage grâce à un compteur, entre 2 et 16 fois, il permet de limiter le nombre d'échantillons par unité de temps et donc d'économiser de l'espace mémoire en fonction de la fréquence d'échantillonnage.
- Mode *bypass* : le *sampler* ne joue pas le rôle de diviseur. Pour éviter l'introduction d'un délai d'échantillonnage inutile, l'entrée et la sortie du sampler sont directement reliées par un fil.

Figure 4.23 : Schéma bloc du *sampler*.

Remarque : Le signal input de n bits qui entre dans le sampler correspond au signal `WPO` sortant des cellules d'observations du WBR. La valeur de `n` est donc égale à la variable générique `n_channel_obs`.

4.4.2.2 Trigger

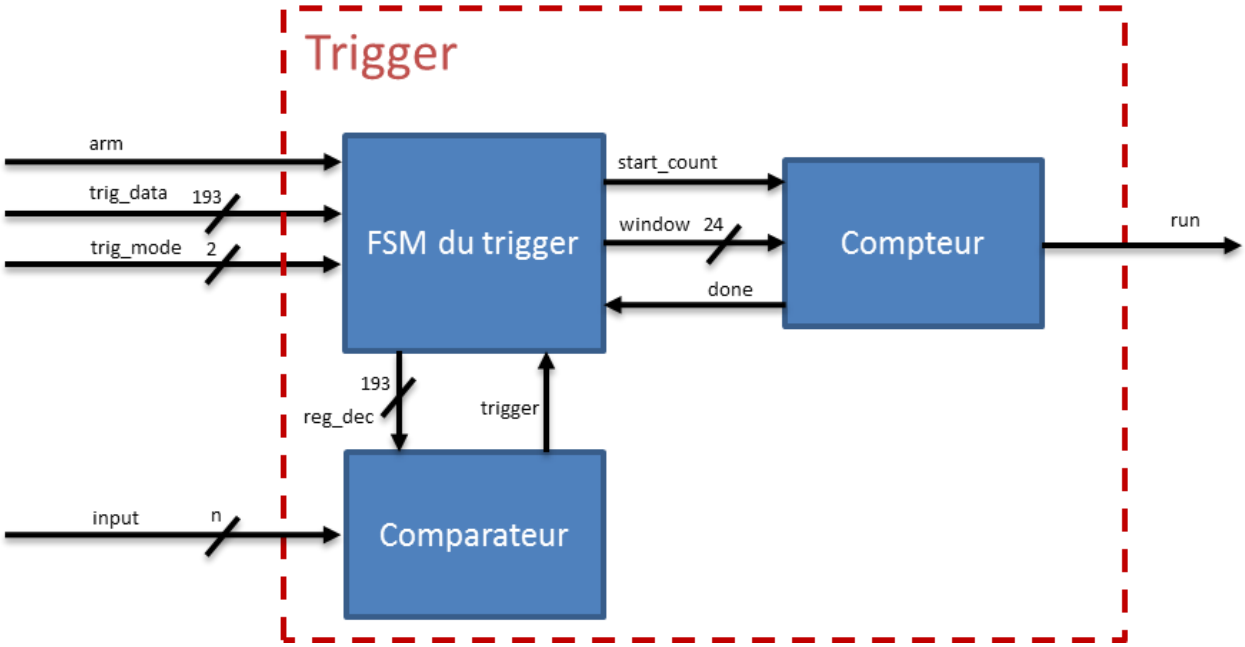


Figure 4.24 : Architecture détaillée du *trigger*.

Le *trigger* est composé de trois sous modules (Figure 4.24) :

- Le bloc comparateur contient la logique qui lui permet de comparer les échantillons d'entrée (sur le signal `input`) avec la condition de déclenchement dont les paramètres sont contenus dans le registre de déclenchement (`reg_dec`).
- Le bloc FSM du *trigger* permet de mettre à jour les registres de déclenchement et WINDOW. Il a aussi pour rôle de gérer les évènements qui permettent l'échantillonnage : mise à feu, déclenchement, début et fin de l'échantillonnage.
- Le bloc compteur permet simplement d'émettre le signal `run` pendant la durée spécifiée par le registre WINDOW

Remarque : l'entrée `input` de ce module correspond au signal `sample` connecté à la sortie du bloc *sampler*.

Chacun de ces sous-modules est présenté avec plus de détails dans la suite.

4.4.2.2.1 Comparateur

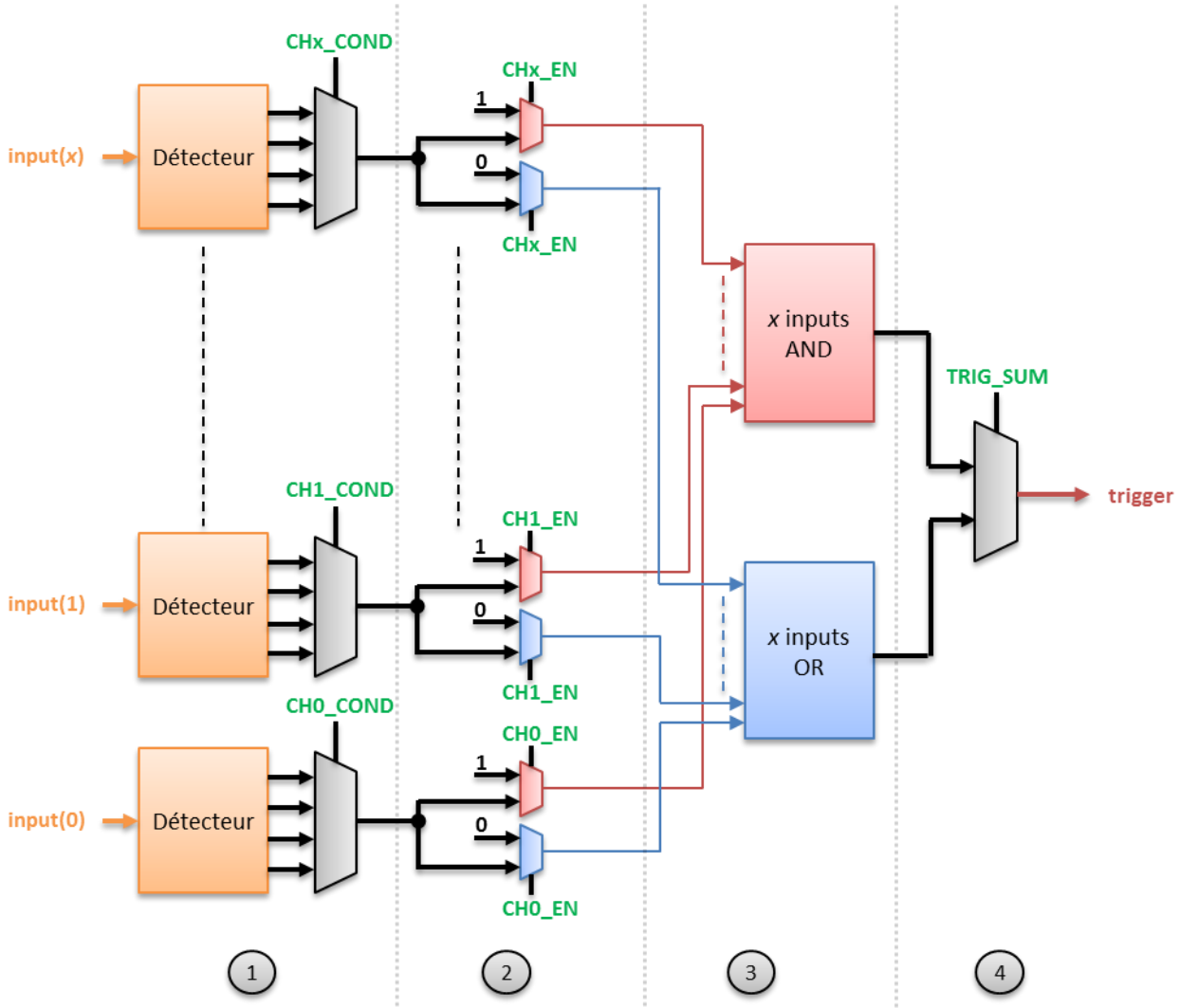


Figure 4.25 : Architecture détaillée du comparateur (générique).

L'architecture détaillée du comparateur, générique, est présentée sur la Figure 4.25. On considère un nombre *x* d'entrées, qui correspond à la variable générique `n_channel_obs`, le nombre de canaux d'observation. L'architecture en quatre étages combinatoires est détaillée :

1) Bloc de détection : chaque entrée est dirigée vers un bloc détecteur (Figure 4.26) qui donne quatre informations de l'état de cette entrée (1 logique, 0 logique, front montant et front descendant). Un multiplexeur permet ensuite de sélectionner l'information sur l'entrée concernée en fonction de la condition « CHx_COND » contenue dans le registre de déclenchement.

2) Bloc de *bypass* : l'information sur l'entrée entre dans deux multiplexeurs de bypass. Si l'entrée n'est pas activée pour faire partie de la condition de déclenchement (« CHx_EN », registre de déclenchement), alors l'information sur l'entrée est évitée (*bypass*). La valeur de *bypass* est 0 ou 1 en fonction de la destination dans le bloc suivant.

3) Bloc de somme des conditions : il y a deux fonctions logiques, ET et OU pour sommer les informations sur les entrées (conditions). Concernant les valeurs de *bypass*, pour la fonction ET, un '1' logique est transparent et pour la fonction OU, un '0' logique est transparent.

4) Bloc de sélection de somme : un dernier bloc permet de choisir la somme des conditions en fonction du signal « TRIG_SUM » du registre de déclenchement.

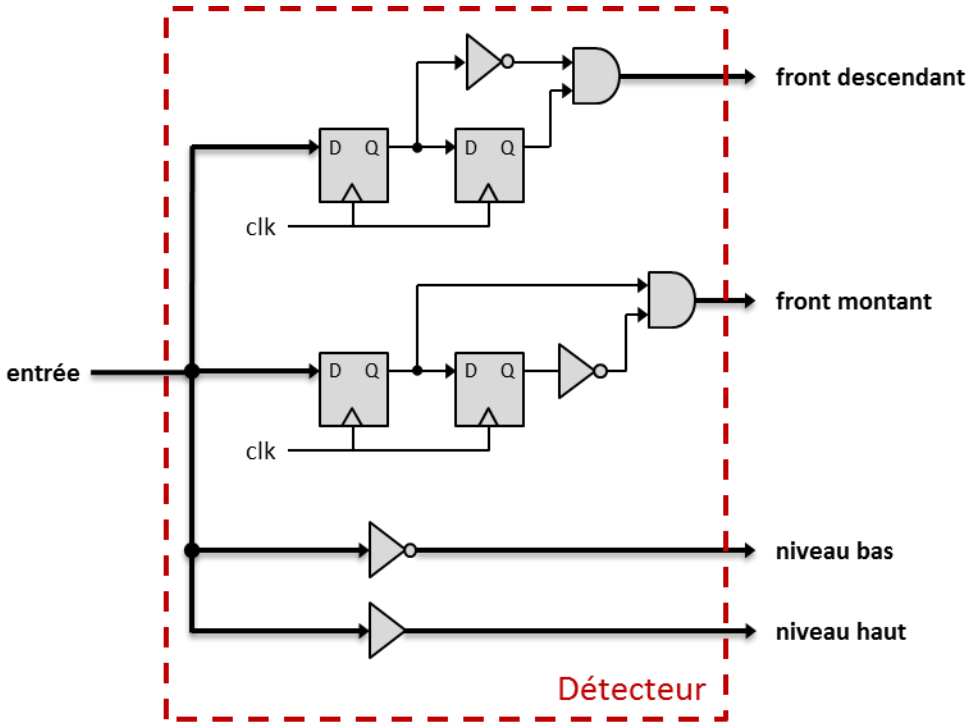


Figure 4.26 : Architecture détaillée du détecteur.

4.4.2.2.2 Machine à états finis du trigger

Le diagramme d'état de la machine à états du trigger est présenté sur la Figure 4.27. Les opérations dépendent du signal `trig_mode` :

- `trig_mode`=01 : le registre de déclenchement `reg_dec` est mis à jour

- `trig_mode`=10 : le registre `window` est mis à jour
- `trig_mode`=00 : gestion des évènements liés à l'échantillonnage :
    - Attente du front montant du signal `arm` : mise à feu
    - Protection si `window`=0, il n'y a pas d'échantillonnage
    - Attente du signal `trigger`, détection de la condition de déclenchement : début du comptage (`start_count`)
    - Attente du signal `done`, fin de la période d'échantillonnage : arrêt du comptage.

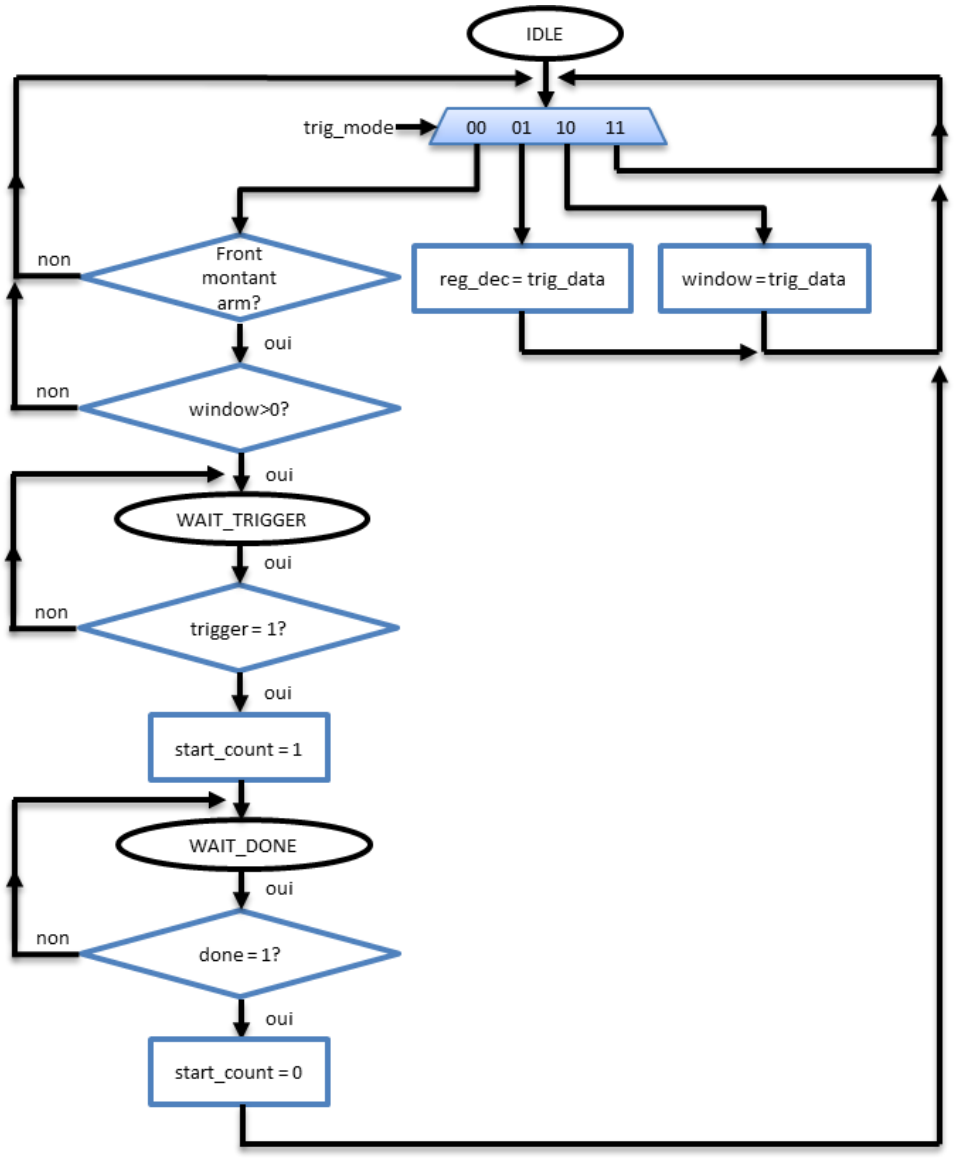


Figure 4.27 : Diagramme d'états de la machine à états finis du trigger.

Pendant l'échantillonnage, dès la détection d'un front montant du signal `arm`, il n'est plus possible de changer les conditions de déclenchement ou la taille de la période d'échantillonnage.

4.4.2.2.3 Compteur

Ce module est un simple compteur activable grâce au signal `start_count`. Quand il atteint la valeur contenue dans le registre `window`, il affecte le signal `done` à 1.

4.5 Module contrôleur, spécifications et architecture

4.5.1 Spécifications du contrôleur

Le bloc contrôleur est un module qui enregistre ou lit des mesures de l'analyseur logique (via le bus `sample`) dans une mémoire externe de type DDR2. Un bloc d'interface physique généré par Xilinx Core Generator est utilisé : le MCB (*Memory Controller Bloc*). Ce dernier gère les opérations de communication avec la mémoire externe. Il est contrôlé par des FIFOs (*First In First Out*) de commandes et de données qui permettent d'échanger des données entre la mémoire et le reste du système à des fréquences différentes (Figure 4.28). En résumé, l'objectif du module contrôleur est de s'interfacer avec les différentes FIFOs du MCB de Xilinx.

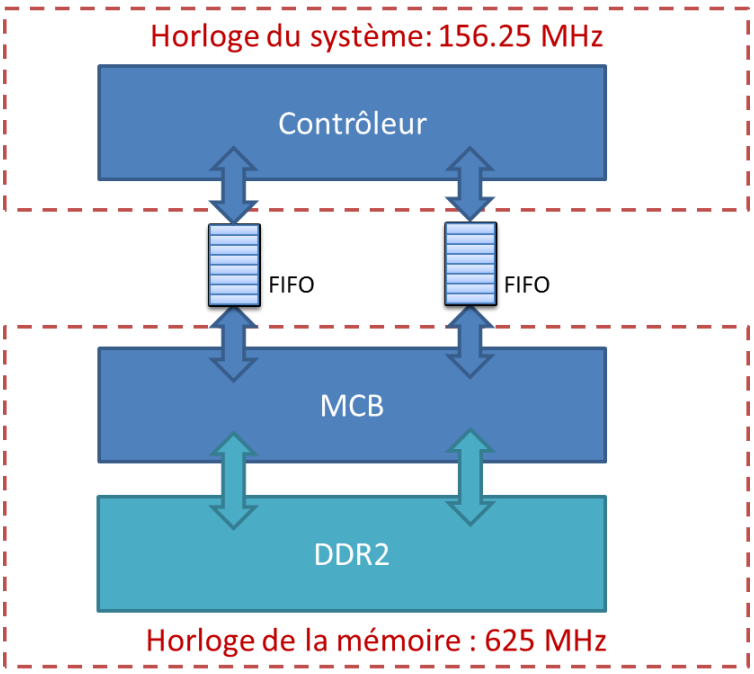


Figure 4.28 : Interface mémoire à plusieurs domaines d'horloge.

4.5.1.1 Bloc MCB (*Memory Controller Bloc*)

On présente un résumé du principe de fonctionnement du MCB (Figure 4.29), ainsi que les caractéristiques choisies pour ce design. Toutes les informations proviennent de la fiche technique du MCB [43].

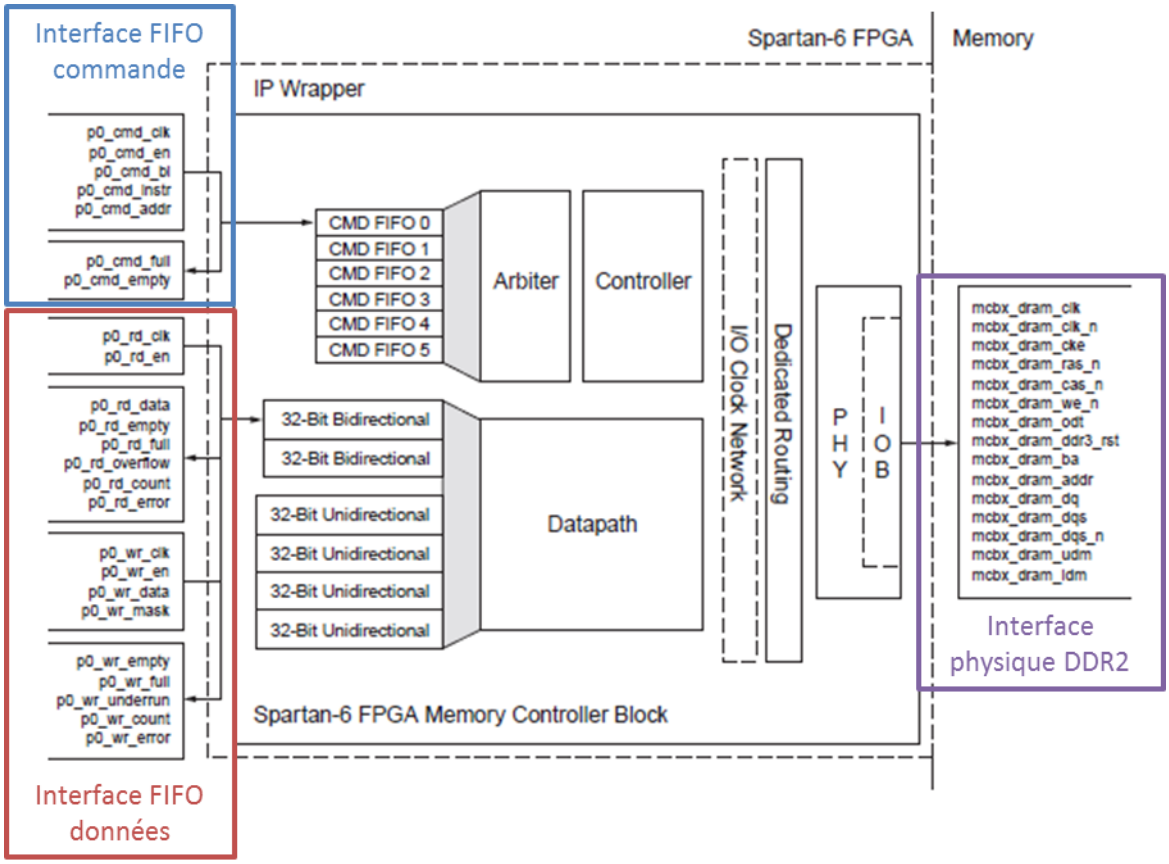


Figure 4.29 : Schéma d'exemple d'un MCB (*Memory Controller Bloc*) obtenu avec Core Generator (tiré de [43]).

Le bloc MCB joue le rôle d'intermédiaire entre l'interface physique DDR2 (signaux à droite de la figure) et l'interface de contrôle de l'utilisateur (signaux à gauche de la figure). Le MCB dispose d'un nombre de ports de données variable en fonction des applications (bande passante de 128 bits bidirectionnels), le choix s'est tourné vers deux ports bidirectionnels de 64 bits. La taille des ports de données a été choisie pour permettre jusque 64 canaux en entrée tout en respectant les spécifications de l'ensemble des modules (1 bit par canal). Dans cette version, un seul port sera utilisé pour la lecture et l'écriture. L'objectif à long terme est d'utiliser un port en entrée et l'autre en sortie afin de permettre la lecture et l'écriture simultanée en mémoire. Le MCB dispose d'un port de commande pour chaque port de donnée. Enfin, tous les ports sont des FIFOs (FIFO

de commande ou FIFO de donnée) pour permettre un fonctionnement à deux domaines d'horloge.

4.5.1.1.1 Principe de fonctionnement du MCB de Xilinx

Le principe de fonctionnement du MCB de Xilinx, tel qu'il est utilisé, est simple. Le module attend qu'une commande soit écrite dans la FIFO de commande. Ensuite, il utilise la FIFO de données comme une source pour une écriture ou comme une destination dans le cas d'une lecture.

4.5.1.1.2 Contrôle des FIFOs de commande

On donne le rôle des signaux principaux de contrôle des FIFOs de commande (« X » désigne l'index du port) :

- `pX_cmd_instr`[2:0], code d'instruction, voir le Tableau 4.15
- `pX_cmd_addr`[29:0], bus d'adresse
- `pX_cmd_bl`[5:0], ce signal indique le nombre de mots manipulés lors d'une opération de lecture ou d'écriture
- `pX_cmd_en`, ce signal permet d'écrire la commande dans la FIFO de commande

Tableau 4.15 : Instructions des FIFOs de commande du MCB de Xilinx.

Instruction	**`pX_cmd_instr`[2:0]**	**Description**
Écriture	**000**	Écriture du nombre de mots spécifié dans le signal `pX_cmd_bl`[5:0] à partir de l'adresse spécifiée par `pX_cmd_addr`[29:0]
Lecture	**001**	Lecture du nombre de mots spécifié dans le signal pX_cmd_bl[5:0] à partir de l'adresse spécifiée par pX_cmd_addr[29 :0]
Écriture avec précharge	**010**	Non utilisé. Instruction similaire à l'écriture, mais le MCB optimise la gestion de la mémoire pour les accès aléatoires.

Lecture avec précharge	**011**	Non utilisé. Instruction similaire à la lecture, mais le MCB optimise la gestion de la mémoire pour les accès aléatoires.
Rafraichissement	**1XX**	Non utilisé. Rafraichissement commandé (non automatique) de la mémoire.

On donne un exemple d'écriture de commande dans la FIFO de commande sur la Figure 4.30.

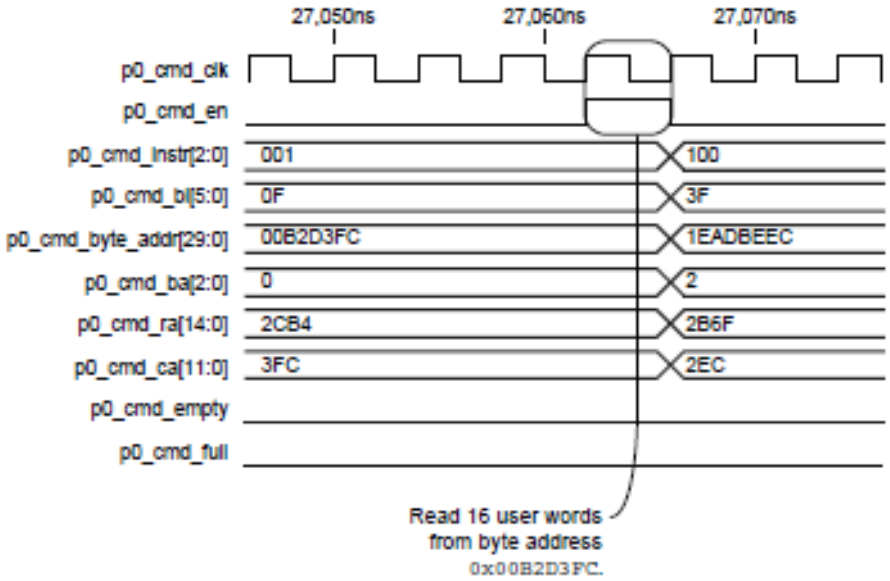


Figure 4.30 : Exemple du déroulement temporel de l'écriture d'une commande de lecture dans la FIFO de commande (tiré de [43]).

4.5.1.1.3 Contrôle des FIFOs de données

On donne le rôle des signaux principaux de contrôle des FIFOs de données (« X » désigne l'index du port) :

- `pX_wr_data`[63:0], bus d'entrée de données (écriture) de la FIFO de données
- `pX_wr_en`, ce signal permet d'écrire une donnée dans la FIFO de données
- `pX_rd_data`[63:0], bus de sortie de données (lecture) de la FIFO de données
- `pX_rd_en`, ce signal permet de lire une donnée dans la FIFO de données

Les bus de données sont de la taille choisie pour les ports, 64 bits dans notre cas. Les Figure 4.31 et Figure 4.32 donnent des exemples d'écriture et de lecture dans la FIFO de données.

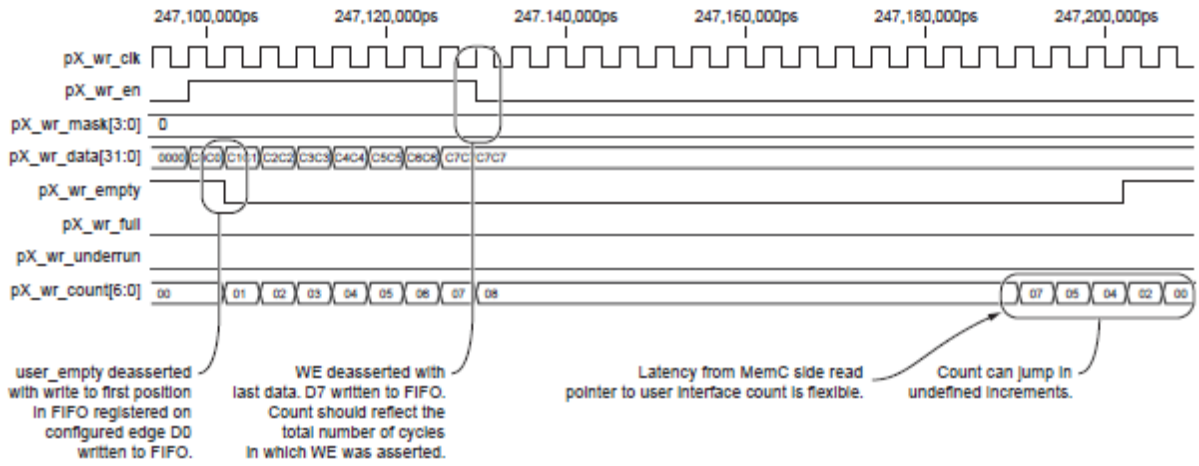


Figure 4.31 : Exemple du déroulement temporel de l'écriture de données dans la FIFO de données (tiré de [43]).

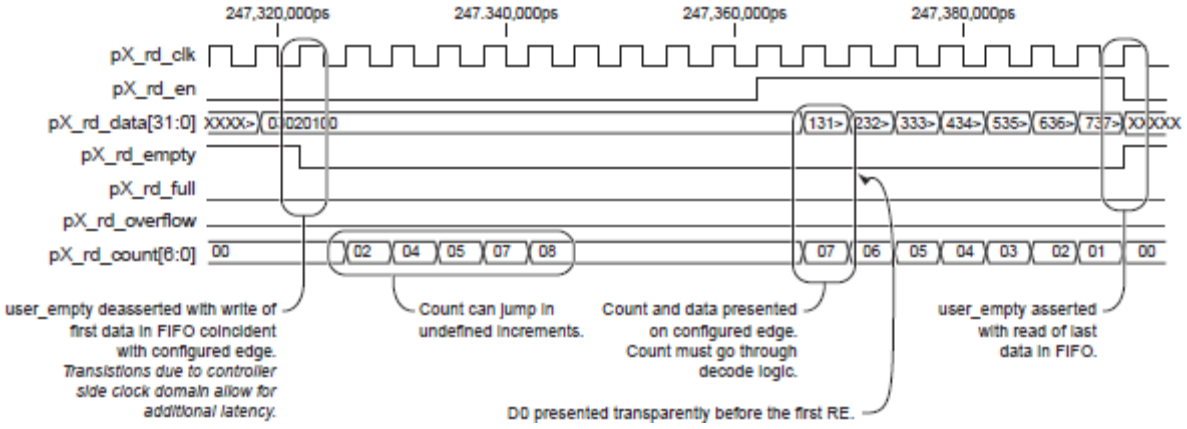


Figure 4.32 : Exemple du déroulement temporel de la lecture de données dans la FIFO de données (tiré de [43]).

4.5.1.1.4 Caractéristiques de la mémoire DDR2

Les caractéristiques de l'interface et de la mémoire DDR2 sont les suivantes :

- Mémoire Micron MT47H64M16-25 de 1 Gbit DDR2 (64 millions d'adresses de 16 bits de données)
- Fréquence de la mémoire : 625 MHz
- 2 ports de données bidirectionnels de 64 bits

La fréquence maximale de la mémoire DDR2 est de 800 MHz.

4.5.1.2 Bloc Contrôleur

Le bloc contrôleur a pour objectif d'interfacer le système avec le MCB, et donc la mémoire externe. Deux signaux permettent de contrôler le bloc contrôleur : `rw` et `run` (Tableau 4.16).

Tableau 4.16 : Modes de fonctionnement du bloc contrôleur

rw	run	Description
0	**0**	Idle
0	**1**	**Écriture** : copie des données du bus d'entrée `sample` (n bits) vers la mémoire. Se termine quand `run` passe à 0 ou si la mémoire est pleine.
1	**0**	**Lecture** : lecture de toutes les données qui ont été écrites en mémoire depuis la dernière lecture. Copie des données lues en mémoire sur le bus de sortie `mem_data` (n bits).
1	**1**	Idle

Remarque : le contrôleur ne permet pas une lecture et une écriture simultanée dans ce projet. La variable n correspond à la variable générique `n_channel_obs`.

Les ports de lecture/écriture du MCB sont de 64 bits. Le contrôleur doit alors attendre plusieurs séries de données du bus d'entrée (ou de la FIFO de données pour la lecture) avant de remplir 64 bits. Ce nombre dépend du nombre de canaux d'observations `n_channel_obs`. On a :

- Nombre de séries pour remplir 64 bits = 64 / `n_channel_obs` (division entière)
- Nombre de bits inutiles par paquet de 64 bits = 64 % `n_channel_obs` (reste). Plus cette valeur est grande et plus l'espace mémoire inutilisé sera grand, car les informations ne sont pas fragmentées entre les paquets de 64 bits.

Exemple avec **20** canaux d'observation (Figure 4.33) :

- Nombre de séries pour remplir 64 bits = 64 / **20** = 3 donc 3 cycles seront nécessaire pour envoyer ou pour lire un paquet de 64 bits

- Nombre de bits inutiles par paquet de 64 bits = 64 % **20** = 4 (reste) donc les 4 derniers bits de chaque paquet écrit ou lu sera toujours inutile

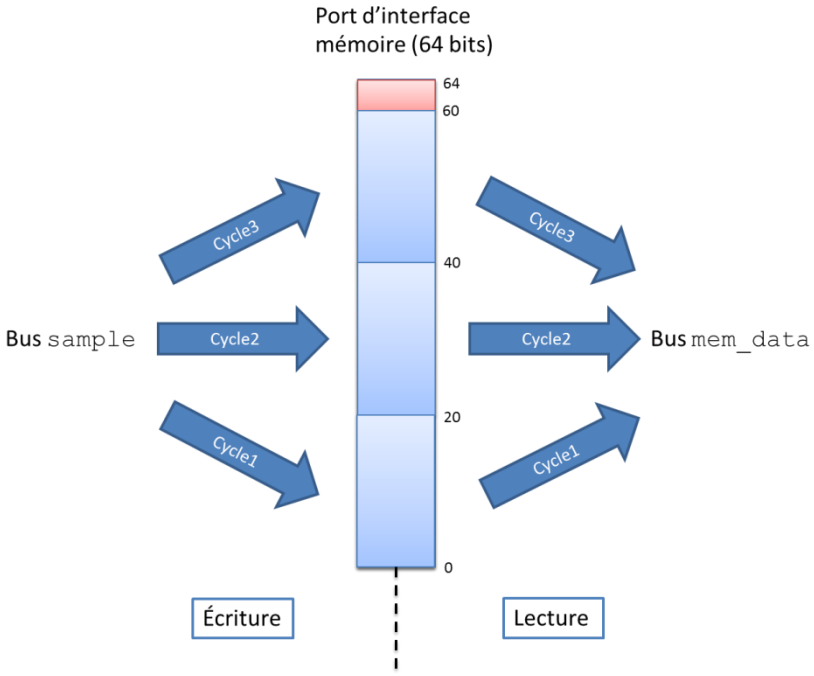


Figure 4.33 : Schéma de l'interface des données entre les bus sample/mem_data (n bits) et le port de données du MCB (64 bits) — Exemple avec n=20.

4.5.2 Architecture du contrôleur

L'architecture du module contrôleur est représentée sur la Figure 4.34. Quelques signaux de contrôles n'y sont pas représentés pour permettre plus de clarté. Le contrôleur dispose de deux accumulateurs de 64 bits, un en entrée (WR_ACC) et un en sortie (RD_ACC), qui lui permettent de passer d'un bus de n bits vers 64 bits, ou inversement. Il y a aussi un sous-bloc pour répondre aux commandes de lecture ou d'écriture de l'utilisateur (FSM du contrôleur) et un sous-bloc permettant de gérer les commandes envoyées au MCB (command writer). Les Figure 4.35, Figure 4.36, Figure 4.37 et Figure 4.38 donnent les diagrammes d'états correspondants au fonctionnement de ces sous-blocs. La mise en place de deux machines à états finis plutôt qu'une seule a été pensée dans l'optique d'ajouter la fonctionnalité de lecture/écriture simultanée. De plus, cela permet une meilleure compréhension de la logique. Une machine à états finis s'occupe du chemin de donnée, l'autre gère les commandes au MCB.

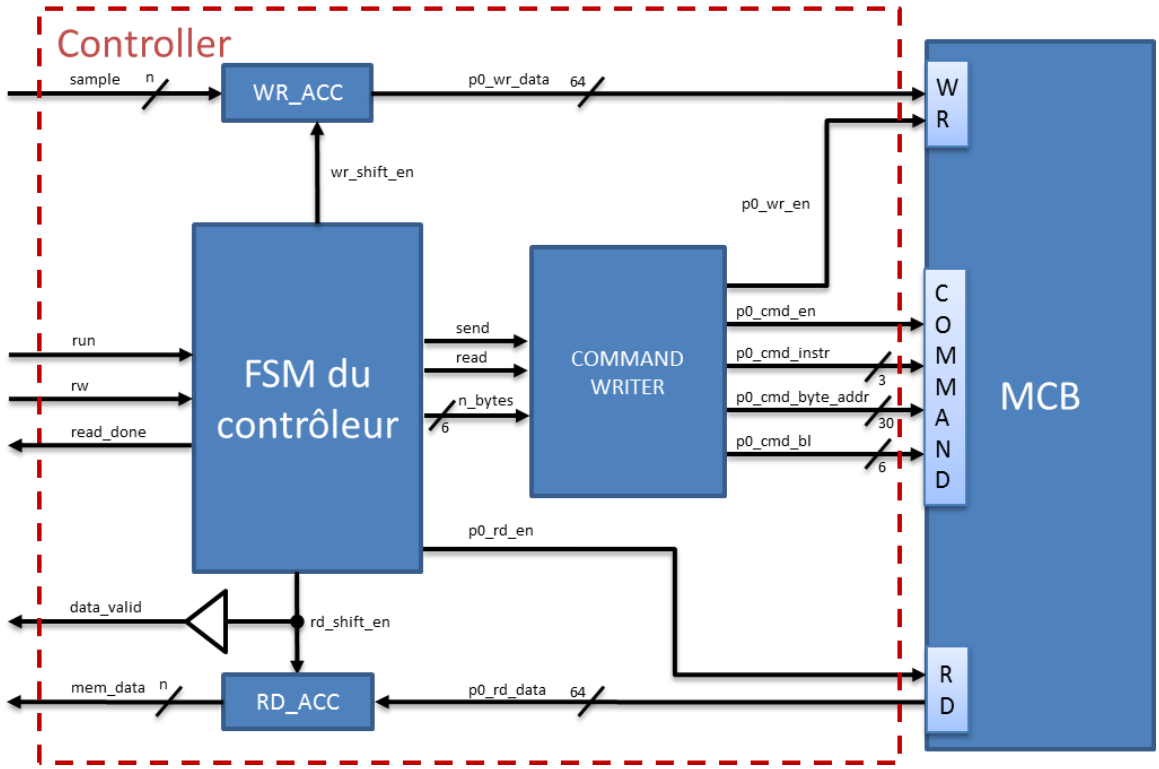


Figure 4.34 : Architecture détaillée du contrôleur.

4.5.2.1 Machine à états finis du contrôleur

Ce sous-bloc répond aux signaux de contrôle `rw` et `run`, pour déclencher les opérations de lecture et d'écriture (Figure 4.35).

Dans le cas de l'écriture (Figure 4.36), des paquets de 64 bits sont envoyés en mémoire tant que c'est demandé. Le signal générique `n_data_in_64b` sert à savoir combien de vecteurs d'entrée de données peuvent être contenus dans l'accumulateur de 64 bits. Le signal `send` permet de demander une écriture au module command_writer.

```
n_data_in_64b = 64 / n_channel_obs
```

Pour la lecture (Figure 4.37), la mémoire est lue par séries de 64 paquets de 64 bits puis chaque paquet est placé dans l'accumulateur de sortie pour ressortir via le signal `mem_data`. En général la dernière lecture ne contient pas exactement 64 paquets, cette valeur est automatiquement calculée et est gérée par le module d'écriture de commandes (command writer). Il y a la variable générique `n_data_in_64b` qui permet cette fois-ci de décomposer un paquet en un nombre

adapté de vecteurs (Figure 4.33). Il y a aussi une variable interne, *N*, qui contient le nombre de commandes au MCB nécessaires à la lecture de tous les paquets non lus en mémoire. Le signal `read` permet de demander une lecture au module command_writer.

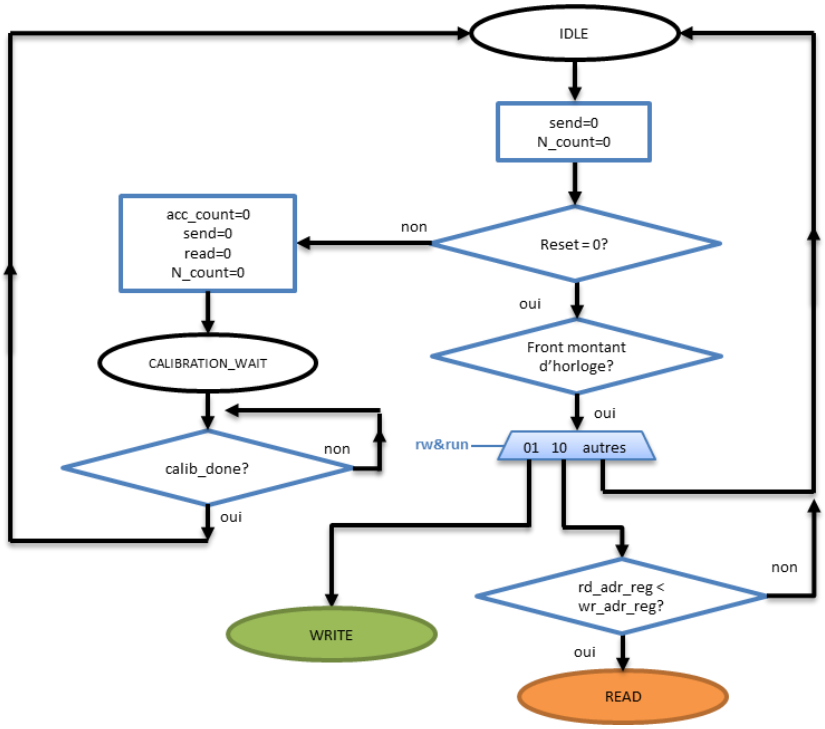


Figure 4.35 : Diagramme d'état de la machine à états finis du contrôleur – 1ère sur 3 (principal).

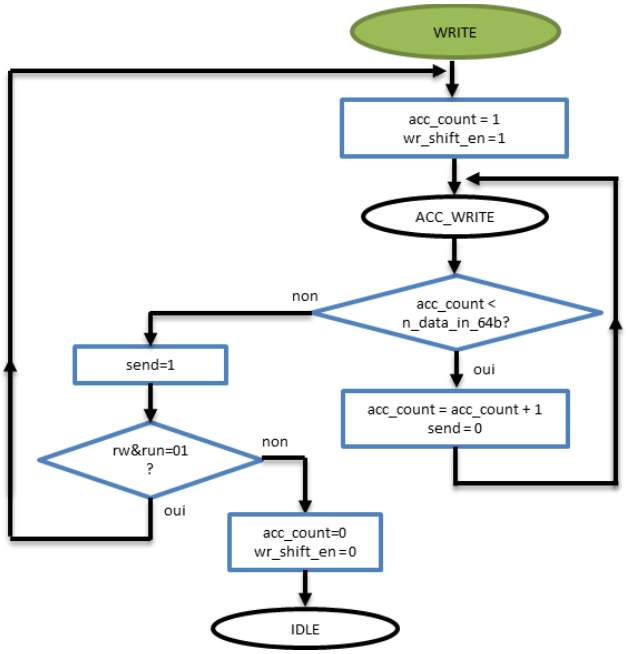


Figure 4.36 : Diagramme d'état de la machine à états finis du contrôleur – 2ème sur 3 (écriture).

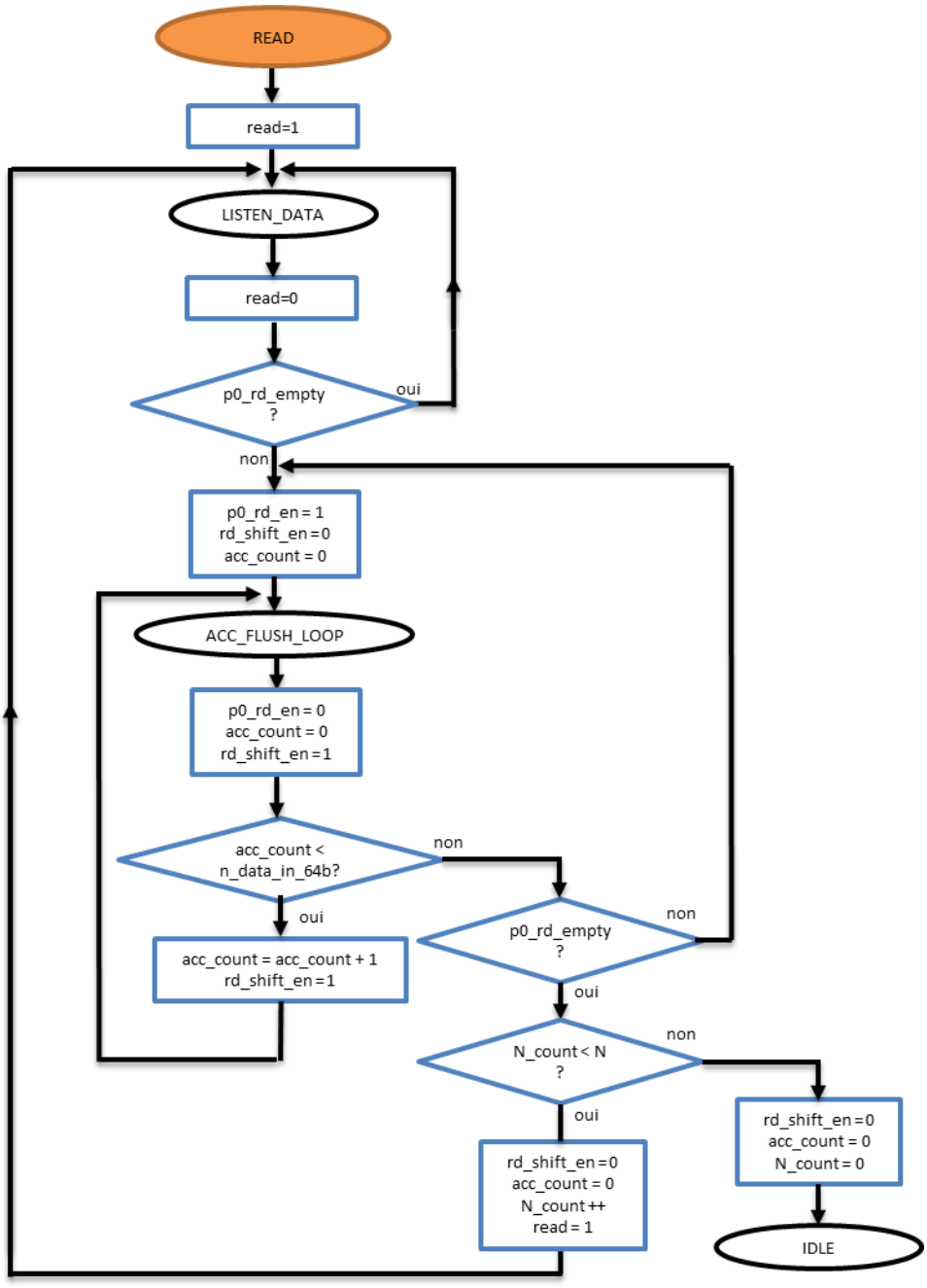


Figure 4.37 : Diagramme d'état de la machine à états finis du contrôleur – 3ème sur 3 (lecture).

4.5.2.2 *Command writer*

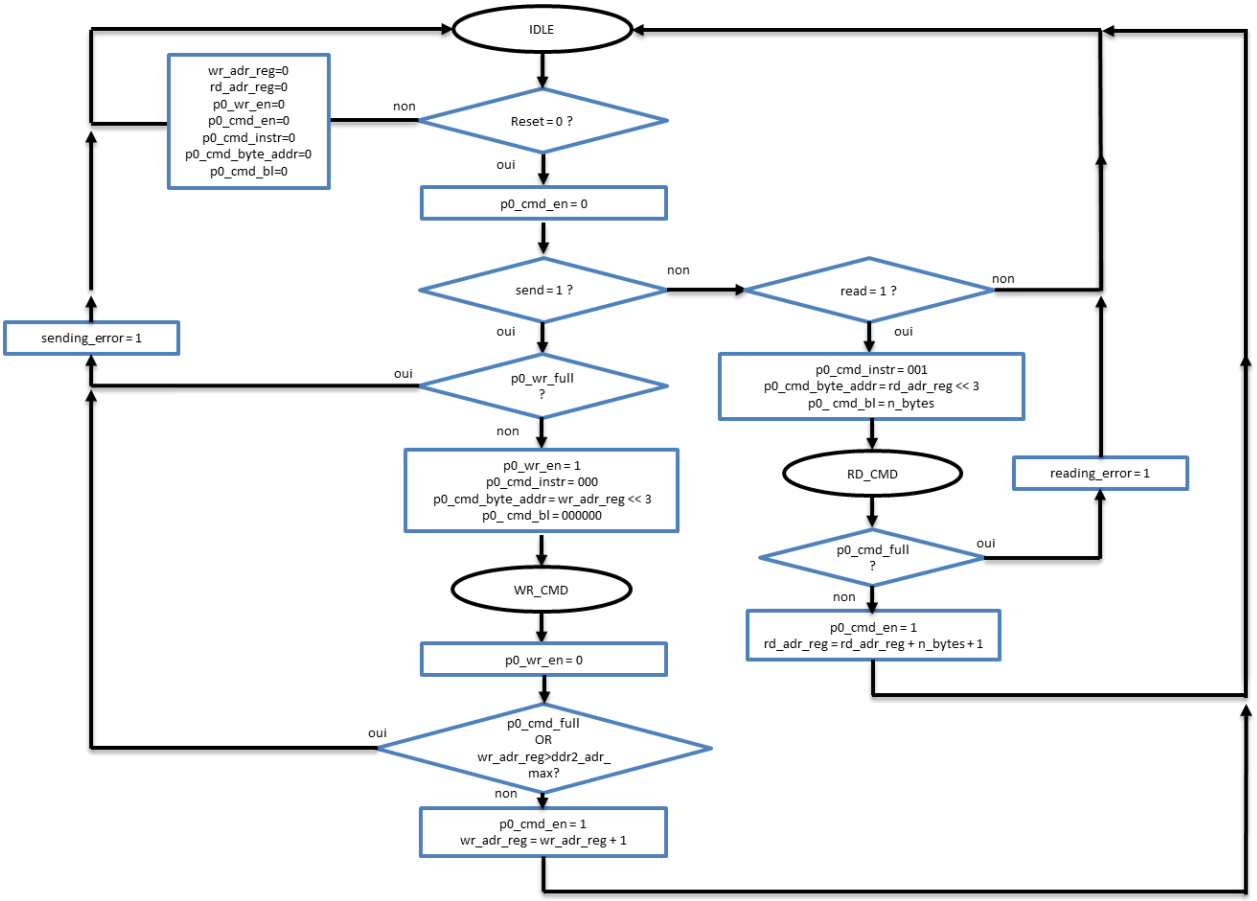


Figure 4.38 : Diagramme d'état de la machine à états finis du module command writer.

Le sous-bloc d'écriture des commandes au MCB (Figure 4.38) s'occupe de l'écriture des informations dans la FIFO de commande. Il met également à jour la valeur des adresses d'écriture et de lecture. Une ébauche de gestion d'erreur a été imaginée pour les erreurs de communication avec le MCB, mais n'est pas fonctionnelle dans cette version.

4.6 Bilan sur l'architecture de la solution proposée

Ce chapitre a répondu au besoin de construire un circuit in-situ et reconfigurable pour la vérification de systèmes électroniques, compatible avec le WaferBoard™. La solution proposée a été d'implémenter une architecture DFT, pour FPGA, donnant les fonctionnalités nécessaire à la vérification. Cette architecture est principalement constituée des modules suivants :

- Une enveloppe DFT basée sur la norme IEEE 1500 permettant de créer des cellules d'observation ou de contrôle des signaux aux entrées/sorties des circuits intégrés qui composent le système.
- Un analyseur logique qui permet de choisir des conditions particulières d'observation des signaux.
- Un contrôleur de mémoire pour pouvoir interfacer une mémoire externe et ainsi augmenter la quantité de données qui peuvent être stockées dans l'outil.
- Un décodeur qui configure et contrôle les trois modules précédents en fonction d'instructions utilisateurs reçues en entrée de l'outil.

Cette solution est adaptée au contexte du WaferBoard™ puisqu'elle propose un outil, déposé in-situ sur le WaferIC™, qui peut se mêler à d'autres circuits intégrés du système électronique prototypé. De plus, son caractère reconfigurable lui permet d'être modifiée dynamiquement pour répondre aux besoins de vérification d'un utilisateur de la plateforme WaferBoard™. Le prochain chapitre présentera la mise en œuvre de l'outil de vérification sur une carte de développement. Son objectif est de caractériser et de quantifier les performances de la solution proposée tout en offrant une preuve de concept.

CHAPITRE 5 PREUVE DE CONCEPT

Dans ce chapitre est étudiée la mise en œuvre de l'outil présenté dans le Chapitre 4, Outil de vérification proposé, page 51. Ainsi, une preuve de concept est constituée, soutenant l'architecture et le design de l'outil avec un ensemble de résultats et de caractéristiques mesurables. D'abord, l'environnement de test qui a permis d'étudier le système sera présenté. L'objectif est d'implémenter l'outil sur une carte de développement FPGA pour pouvoir interagir avec lui. Ensuite, seront détaillés l'ensemble des résultats obtenus dans le cas d'un exemple de système, un microcontrôleur PIC, sous vérification. Enfin, le design sera étudié avec plus de finesse pour fournir une caractérisation des performances à travers différents cas de tests.

5.1 Environnement de test

L'environnement de test doit être décrit avec précision puisque c'est à travers lui que les résultats de test d'un circuit sont recueillis. Il correspond à la nature et les caractéristiques du matériel utilisé, mais aussi à la configuration des circuits mis en œuvre et l'ensemble des vecteurs de test appliqués. De plus, il contient des méthodes de mesures et d'analyse des résultats. Finalement, l'environnement de test doit être suffisamment détaillé pour minimiser les erreurs et les incertitudes qui incombent à la vérification et à la validation d'un design.

5.1.1 Carte de développement FPGA

L'outil de vérification a été implémenté en VHDL sur un FPGA Spartan6 LX45, monté sur la carte de développement ATLYS de Digilent. Cette dernière dispose des ressources pertinentes suivantes :

- Mémoire DDR2 Micron MT47H64M16-25E de 1 Gbit ou 64 millions de cases de 16 bits compatible avec des taux de transfert jusqu'à 800 MHz : cette mémoire est compatible avec les MCB (Memory Controller Block) de Xilinx, elle permet de proposer un système avec une plus grande capacité de stockage que la simple mémoire interne d'un FPGA.
- Horloge de base à 100 MHz : cette horloge permet de générer les différentes horloges du design, pour la mémoire externe ou pour le cœur du système, à partir d'une PLL. L'horloge principale du système est de 156.25 MHz et l'horloge de la mémoire est de

625 MHz. La section 5.3.4, page 110, donne plus de détails à propos des horloges utilisés dans l'outil.

- Port d'extension de 8 entrées/sorties : ce port permet de faire des tests simples, avec un nombre d'entrées/sorties limitées.
- Port d'extension haute vitesse de 40 entrées/sorties : ce port permet de faire des tests plus complexes.

La suite de logiciel ISE de Xilinx a été utilisée pour la synthèse, le placement, le routage et la création du fichier de configuration.

5.1.2 Chipscope

L'outil ne comporte pas d'interface de communication avec un ordinateur. Ainsi, il n'est pas possible d'envoyer des commandes ou de récupérer des informations lues en mémoire. Pour répondre à ce manque, des circuits logiques Chipscope de Xilinx, ILA et VIO, ont été mis en place. Comme ils ont été décrits dans la section 3.2.2.2, ces outils offrent l'interface nécessaire pour interagir avec le système sous test.

Il est possible d'utiliser le logiciel Chipscope Inserter pour générer les modules de Chipscope. Il n'a pas été utilisé, car il ne permet pas d'implémenter le module VIO, mais seulement l'ILA. L'utilisation de Chipscope Inserter est en général avantageuse, car elle modifie simplement la netlist (liste de connexions) générée par le logiciel ISE pour interfacer un module ILA. Il n'est alors pas nécessaire de modifier le code source VHDL.

La création des circuits VIO et ILA a donc été faite grâce à l'outil Core Generator de Xilinx, puis ils ont été intégrés dans le code VHDL du module principal (*top level*) du design. L'inconvénient de l'intégration manuelle des modules Chipscope dans le code est qu'il faut ajouter des signaux dans les entités des sous-modules pour avoir accès à leurs signaux internes. En effet, le langage standard VHDL (avant la version VHDL 1076-2008) ne permet pas de désigner un signal par un chemin hiérarchique à travers différentes instances de composants. Il est néanmoins possible d'utiliser des fonctionnalités, comme « *nc* mirror » de Cadence, qui surpassent cette limite.

5.1.2.1 Module de contrôle VIO (*Virtual Input Output*)

Le module VIO est interfacé avec les signaux de contrôle de l'outil de vérification, c'est-à-dire les différents signaux d'entrée qui permettent d'envoyer des instructions au système. C'est la sortie synchrone avec l'horloge du système qui est utilisée.

5.1.2.2 Module d'observation ILA (*Integrated Logic Analyzer*)

Le module ILA est connecté avec la plupart des signaux internes du module principal (*top level*) qui sont utilisés pour relier les sous-modules les uns avec les autres. Les signaux qui sont le plus souvent utilisés pour déclencher l'acquisition des signaux par Chipscope sont présentés :

- `run` : le signal `run` effectue un front montant quand le trigger de l'outil de vérification s'active, il permet donc d'observer ce qu'il se passe au moment où l'acquisition commence.
- `rw` : le signal `rw` effectue un front montant quand une instruction de lecture en mémoire est envoyée à l'outil de vérification. Il est ainsi possible d'observer si les données lues en mémoire correspondent bien aux signaux écrits auparavant.
- `exec` : le signal `exec` effectue un front montant quand une instruction est envoyée à l'outil de vérification

5.2 Étude avec un microcontrôleur PIC

Afin de pouvoir vérifier le fonctionnement du prototype de l'outil réalisé, un périphérique de test, un microcontrôleur, est utilisé pour valider les fonctionnalités de contrôle et d'observation de signaux. Ce dernier représente un exemple de système électronique sous vérification. Dans cette partie est détaillé un cas de vérification particulier dans lequel des trames de données suivant le protocole SPI (*Serial Peripheral Interface*) sont contrôlées et observées. Cet objectif implique la configuration de l'outil de vérification puis une phase d'écriture et de lecture en mémoire. La Figure 5.1 donne un schéma bloc global de l'environnement.

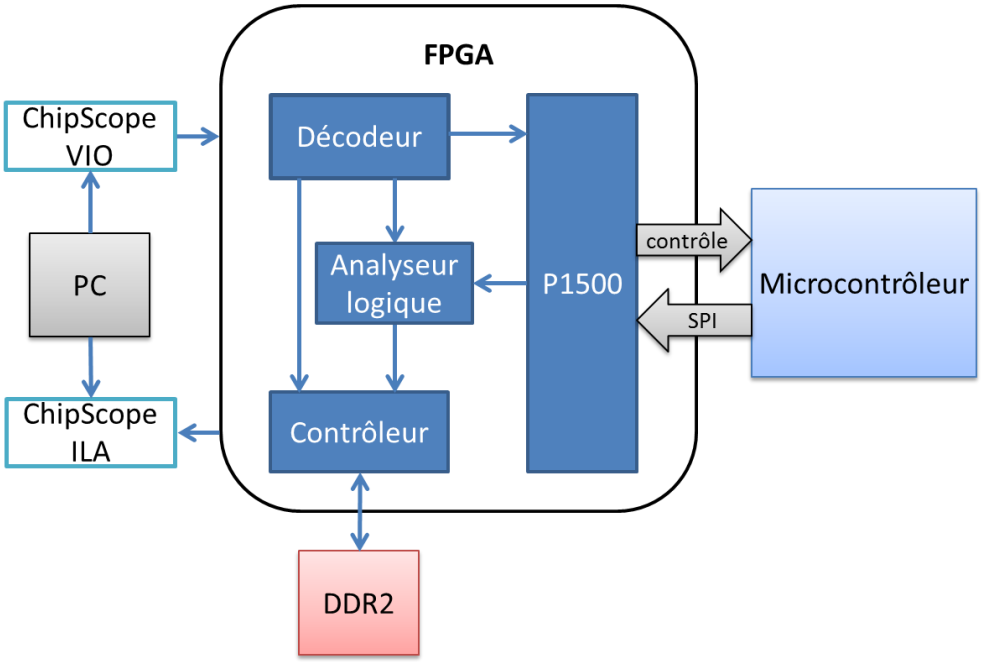


Figure 5.1 : Schéma bloc global de l'environnement de vérification du microcontrôleur PIC.

5.2.1 Microcontrôleur PIC24H

Le système sous vérification choisi est un microcontrôleur de Microchip PIC24H sur une carte de développement. L'intégration d'un microcontrôleur à l'avantage de permettre d'effectuer facilement des tests sur des protocoles de communication rapides, car ils sont déjà implémentés sur celui-ci.

Le microcontrôleur utilisé dans le cadre de notre projet est le PIC24HJ256GP610A sur le kit de développement Explorer16 de Microchip. De plus, la carte de prototypage PICtail Plus est utilisée pour avoir accès à l'ensemble des entrées/sorties. Quelques caractéristiques intéressantes du microcontrôleur pour notre projet sont données :

- Deux modules SPI (*Serial Peripheral Interface*) à 4 fils, fréquence maximale de 10 MHz : ces modules permettent d'étudier l'observation d'une communication cadencée à 10 MHz
- Deux modules I2C (*Inter Integrated Circuit*) et UART (*Universal Asynchronous Receiver Transmitter*) : ces modules permettent d'observer d'autres protocoles à vitesse moindre
- Circuiterie JTAG interne

La suite des travaux détaille le module SPI en particulier, car l'essai de vérification particulier présenté dans cette partie cible cette interface.

5.2.1.1 Configuration du module SPI

Le protocole SPI est un protocole de communication sérielle utilisant quatre signaux. Deux signaux de données nommés SDI (*Serial Data Input*), SDO (*Serial Data Output*), un signal d'horloge nommé SCK (*Shift Clock*) et un signal de synchronisation SS (*Slave Select*).

Un module utilisant le protocole de communication SPI peut avoir deux rôles, maître ou esclave. Le maître est celui qui initie la conversation à l'esclave, c'est-à-dire qu'il contrôle l'horloge et le signal de synchronisation SS.

Dans notre environnement, le PIC sera le maître de la communication. Afin d'observer les messages transmis par le PIC, des cellules d'observation P1500 sont placées sur les lignes du protocole SPI, sorties du PIC : SDO, SCK et SS. Pour vérifier le contrôle de signaux, une entrée du PIC est réservée à l'activation du module SPI. Ce signal, `enable`, permet de contrôler, à partir d'une cellule de contrôle P1500, l'activation de la transmission de données SPI. La Figure 5.2 représente un schéma de l'interface entre le PIC et l'enveloppe P1500 de l'outil de vérification.

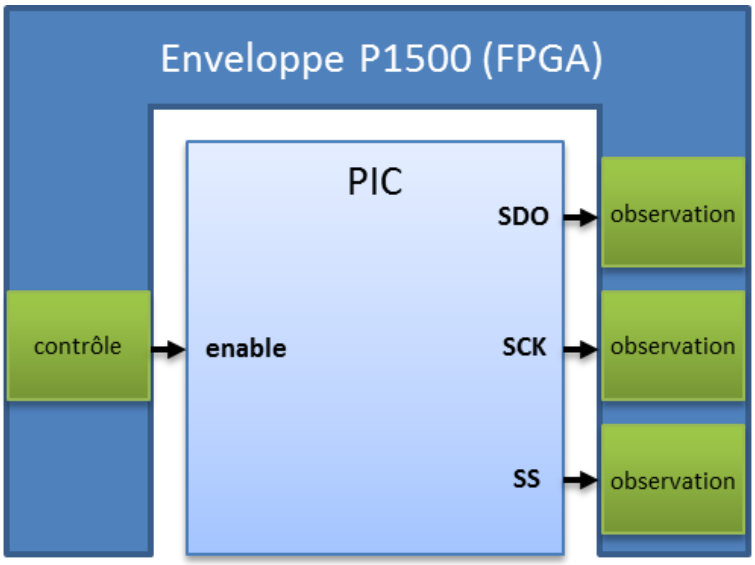


Figure 5.2 : Schéma de l'interface entre le PIC et l'enveloppe P1500 de l'outil de vérification.

5.2.1.2 Paramètres du bus SPI sur le PIC

Les paramètres ci-dessous correspondent à la configuration de module SPI utilisé dans le microcontrôleur : La Figure 5.3 donne une représentation du protocole SPI lors de la transmission d'une donnée sur le bus.

- Niveau bas de l'horloge : repos (*idle*)
- Niveau haut de l'horloge : actif
- Changement des données en sortie sur la transition d'horloge repos vers actif (front montant)
- 10 Mbits/s
- Largeur des mots de données : 8 bits

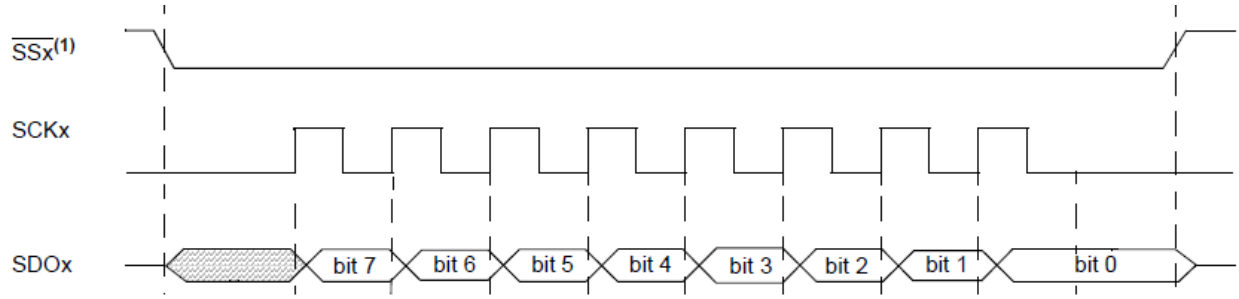


Figure 5.3 : Représentation du protocole SPI lors de la transmission d'une donnée de 8 bits sur le bus.

5.2.1.3 Fonctionnement du programme de gestion de la communication SPI

Le programme du PIC fonctionne de la manière suivante :

```
While (1) {
     If ( enable == 1) SPI_ENVOI ( 0xB5 )
}
```

Tant que le signal **enable** est à 1, le PIC envoie une trame SPI ayant la valeur 0xB5. Une valeur constante facilite la vérification.

5.2.2 Interconnexions

On représente les interconnexions entre les deux cartes de développement du banc d'essai sur la Figure 5.4. Les connecteurs utilisés sont le PICtail Plus du côté de la carte Explorer16 et le connecteur PMOD pour la carte ATLYS. Le connecteur PICtail Plus permet d'avoir un accès

direct à 120 pins du PIC24H. Le connecteur PMOD quant à lui procure une liaison avec 8 pins du FPGA.

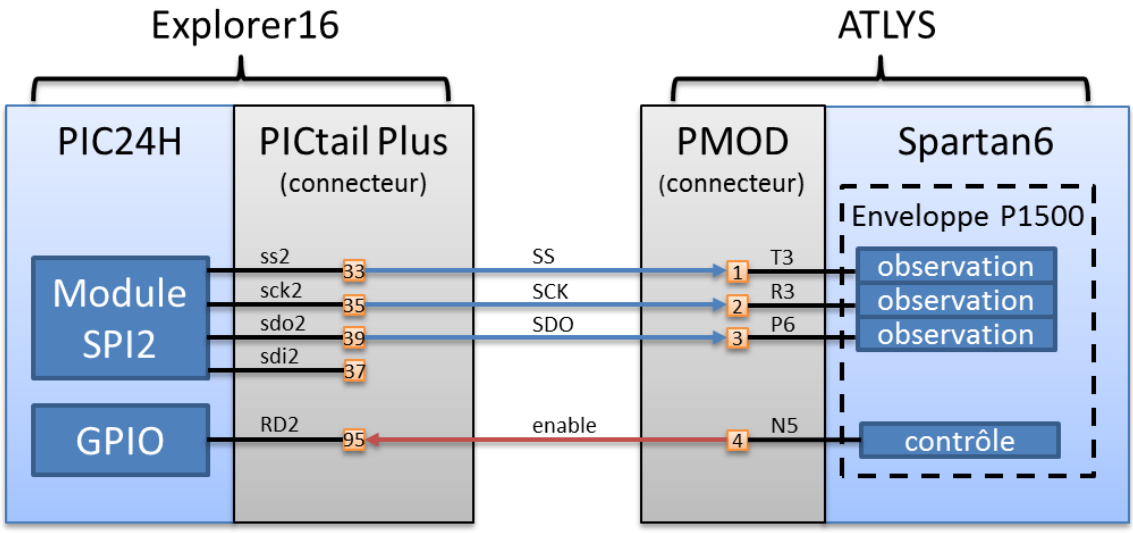


Figure 5.4 : Interconnexions du design de test entre l'outil de vérification (ATLYS) et le périphérique de test (Explorer16).

Il est nécessaire de connecter les signaux de sortie SPI2 du PIC, `SS`, `SCK` et `SDO`, pour les relier aux cellules d'observation dans le FPGA. De plus, le signal de contrôle du FPGA, `enable`, est connecté vers une entrée GPIO du PIC

Les informations concernant les cartes Explorer16 et ATLYS proviennent des documentations officielles (respectivement Explorer 16 Development Card User's Guide et Digilent Inc : Atlys Board Reference Manual).

5.2.3 Configuration de l'outil de vérification

Comme l'outil de vérification possède un caractère reconfigurable, il convient de définir dans quels modes il doit être configuré afin de permettre le déroulement de la vérification du contrôle et de l'observation de trames SPI. De plus, les variables génériques de l'outil sont affectées pour avoir trois cellules d'observations et une cellule de contrôle.

5.2.3.1 Configuration du module P1500

Le module P1500 doit être configuré en mode « WS_EXTEST » (sériel) ou « WP_EXTEST » (parallèle) ce qui permet d'utiliser les cellules de contrôle et d'observation P1500 avec les

entrées/sorties du circuit logique sous vérification, c'est-à-dire le PIC. Les modes sériels et parallèles sont testés.

5.2.3.2 Configuration du module trigger

Le module doit détecter le début d'une trame SPI, or cette dernière commence par une transition descendante du signal SS (Figure 5.3). Le trigger doit donc être configuré pour se déclencher dès la détection d'un front descendant sur le signal SS capturé (`select`). Le but est d'observer au moins une trame SPI circulant dans l'outil de vérification. La durée de cette trame est environ de 11 cycles (Figure 5.3) soit environ 11*100 ns = 1.1µs à 10MHz. Le nombre de cycles à observer à la fréquence du système, 156.25MHz, est donc de 156.25*1.1=172 cycles (arrondi à l'entier supérieur).

Pour une meilleure observabilité, la valeur 255 est choisie.

5.2.4 Résultats

5.2.4.1 Contrôle du signal `enable`

L'objectif de cette série de tests est de vérifier que l'outil permet de contrôler des signaux à partir d'une commande utilisateur. Par cet exemple, la communication SPI en provenance du PIC est activée grâce au contrôle du signal `enable`.

Le Tableau 5.1 donne la liste des instructions envoyées dans l'ordre à l'outil de vérification afin de forcer un « 1 » logique sur la sortie `enable`. Les instructions sont envoyées à travers le module VIO de Chipscope.

Tableau 5.1 : Liste d'instructions envoyées à l'outil de vérification pour contrôler le signal `enable` (configuration parallèle).

Instruction	Data	Description
WIR_SET_MODE	2	Le module P1500 est en mode parallèle
WR_P_SHIFT	1	On applique le vecteur « 0001 » en parallèle aux entrées de test des cellules
WR_UPDATE	--	On copie l'entrée de test sur la sortie fonctionnelle des cellules, dans notre cas `enable` vaut « 1 ». Cette action n'a pas d'effet sur les cellules d'observation.

Après l'envoi de ces trois instructions, une transmission SPI en provenance du PIC est observée, ce qui confirme que le signal `enable` a été mis à « 1 ». La Figure 5.5 donne un aperçu de deux des signaux SPI qui sont générés par le PIC.

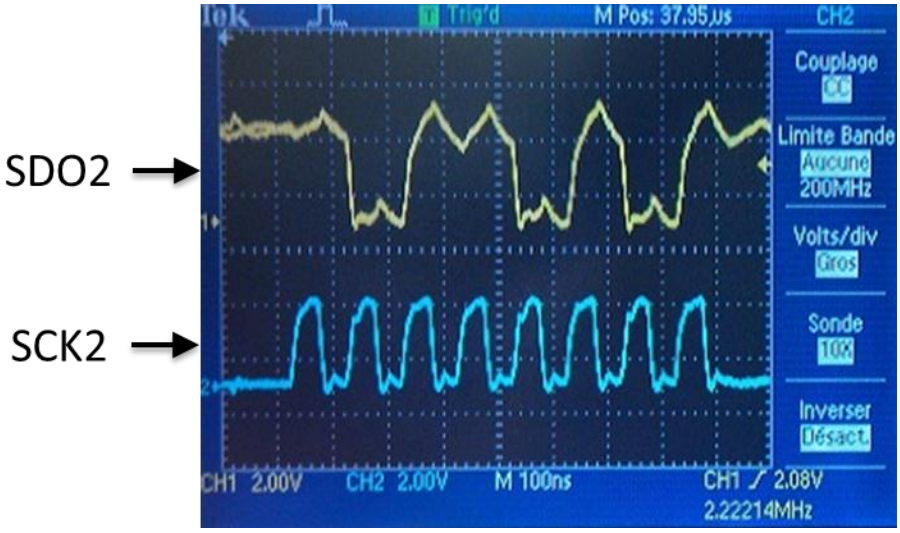


Figure 5.5 : Capture d'écran du signal SPI observé avec l'oscilloscope.

Voie 1 : signal de donnée SDO2 = 0xB5 — Voie 2 : signal d'horloge SCK2 à 10 MHz

Cette capture d'écran valide que l'horloge du SPI est de 10 MHz, et que l'information transmise est 0xB5 en hexadécimal, soit « 1011 0101 » en binaire.

Il est aussi possible de contrôler le signal `enable` en utilisant le mode sériel du module P1500. Le Tableau 5.2 donne les instructions nécessaires.

Tableau 5.2 : Liste d'instructions envoyées à l'outil de vérification pour contrôler le signal `enable` (configuration sérielle).

Instruction	Data	Description
WIR_SET_MODE	1	Le module P1500 est en mode sériel
WR_P_SHIFT	1	On applique « 1 » à l'entrée de test du module 1500. Un seul décalage est nécessaire, car la cellule de contrôle du signal `enable` est la première cellule.
WR_UPDATE	--	On copie l'entrée de test sur la sortie fonctionnelle des cellules, dans notre cas `enable` vaut « 1 ». Cette action n'a pas d'effet sur les cellules d'observation.

5.2.4.2 Observation de la trame SPI

L'objectif de cette série test est de vérifier que l'outil permet d'observer des signaux, de manière sérielle ou parallèle, à partir d'une commande utilisateur. De plus, la fonctionnalité de la mémoire externe est vérifiée, car les données observées sont envoyées dans la mémoire, puis lues et envoyées en sortie de l'outil. Par cet exemple, la communication SPI est échantillonnée puis enregistrée dans l'outil.

Le Tableau 5.3 donne la liste des instructions envoyées dans l'ordre à l'outil de vérification pour pouvoir enregistrer puis lire une trame SPI en mémoire. Les instructions sont envoyées à travers le module VIO de Chipscope.

Tableau 5.3 : Liste d'instructions envoyées à l'outil de vérification pour observer une trame SPI.

Instruction	Data	Description
WIR_SET_MODE	2	Le module P1500 est en mode parallèle
T_DEC_MODE	7	Le trigger est configuré pour se déclencher sur un front descendant du signal échantillonné « select »
T_WINDOW	FF	La largeur de la fenêtre d'écriture en mémoire est de 255
T_ARM	--	Le trigger est armé
WR_CAPTURE	--	Les cellules P1500 capturent les signaux : l'échantillonnage est activé
C_READ	--	Lecture des signaux écrits en mémoire

Pour vérifier le bon fonctionnement de l'outil, le module ILA de Chipscope est utilisé pour observer les signaux internes du FPGA. Les signaux internes utilisés pour le déclenchement de l'ILA sont spécifiés dans la section 5.1.2.2, Interconnexions, page 99 (`run`, `rw` et `exec`).

La Figure 5.6 permet d'observer les signaux internes lors du déclenchement et de l'enregistrement de la trame SPI en mémoire. Une fenêtre de 500 échantillons est sélectionnée avec Chipscope pour permettre d'observer avec en totalité l'échantillonnage de l'outil paramétré à 255 échantillons. Les signaux `sample_x` correspondent aux signaux SPI (`spi_select`, `spi_clock` et `spi_data`) échantillonnés avec un décalage d'un coup d'horloge. Ce décalage correspond aux registres par lesquels passent les signaux dans les cellules d'observation P1500. Le signal `run` passe à 1 quand la condition de déclenchement de l'outil est rencontrée (front

descendant de spi_ss). Il y a ici un décalage de deux coups d'horloges entre la condition de déclenchement et la montée du signal run. Puis le signal run reste à 1 pendant une durée égale au paramètre window, soit 255 cycles. Cette durée est représentée avec les marqueurs O et X sur la forme d'onde de Chipscope. Enfin, le signal read_done passe à 0 dès la montée du signal run, il indique que de nouvelles données sont disponibles en mémoire.

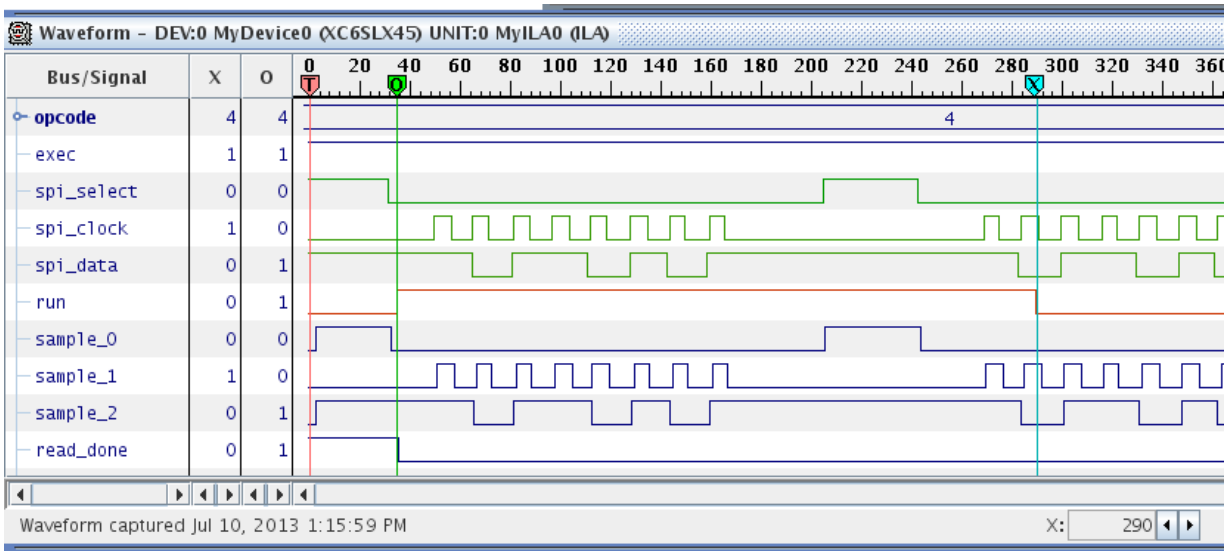


Figure 5.6 : Observation de signaux avec Chipscope : déclenchement du trigger et enregistrement en mémoire.

La Figure 5.7, représente les signaux internes lors de la commande de lecture en mémoire. Les signaux mem_data_x sont les informations lues en mémoire et correspondent aux signaux SPI précédemment échantillonnés (spi_select, spi_clock et spi_data). La lecture continue tant que le signal read_done est à 0. Les données sur le bus mem_data sont valides pour chaque coup d'horloge où le signal data_valid vaut 1. Ce dernier signal passe périodiquement à 0 : ceci correspond à la lecture d'un nouveau paquet de 64 bits en mémoire. En effet, il y a 3 canaux de lecture (mem_data est de largeur 3 bits) donc tous les 21 coups d'horloge, un nouveau paquet de 64 bits doit être lu dans la FIFO de données pour avoir la suite de l'information, avec 21*3 bits = 63 bits (ceci fait référence à la section 4.5.1.2 sur le contrôleur, page 87).

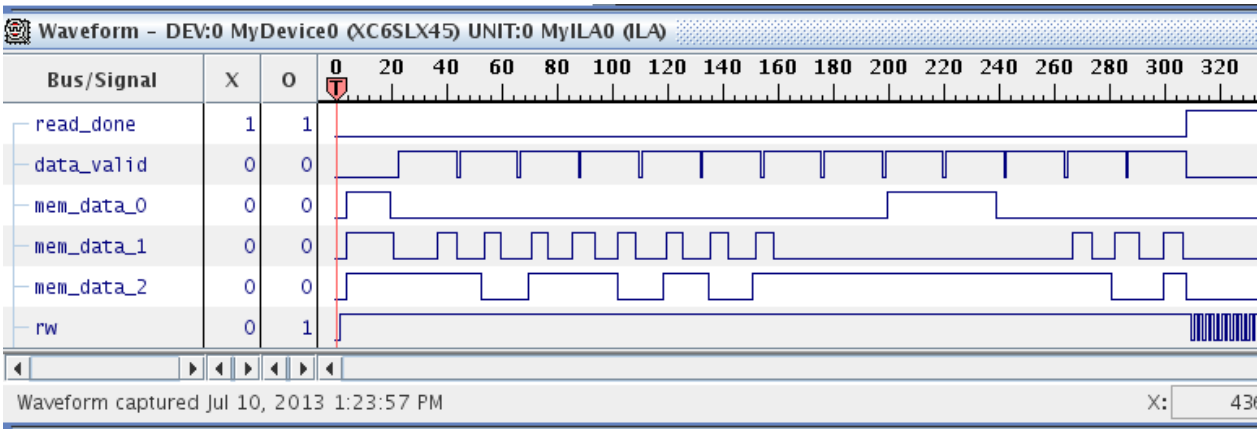


Figure 5.7 : Observation de signaux avec Chipscope : lecture en mémoire.

En conclusion, la trame SPI a correctement été détectée, enregistrée puis restituée en mémoire. Le temps et la condition d'échantillonnage sont respectés. De plus, les instructions demandées dans ce test fonctionnent de la manière attendue.

5.3 Caractérisation des performances

Cette partie présente un ensemble de métriques qui permettent d'évaluer les performances et fonctionnalités du module de vérification. Ce sont les grandeurs caractéristiques et nécessaires pour quantifier les performances d'un outil de vérification :

- Vitesse de traitement des signaux ou bande passante (sections 5.3.2, 5.3.3, 5.3.4, 5.3.5 et 5.3.6) : symbolise la quantité de données qui peuvent être traitée par unité de temps et aussi la vitesse maximale des signaux qui peuvent être contrôlés ou observés.
- Espace de stockage disponible (mémoire, section 5.3.1) : définit la capacité totale de données qui peuvent être enregistrées dans l'outil. Cette grandeur permet d'estimer la quantité de signaux contrôlables ou observable en une série continue et à la vitesse maximale de l'outil.
- Valeurs de comparaison (section 5.3.7) avec d'autres outils ou méthode de vérification de systèmes électroniques. Dans cette section sera notamment étudiée l'accélération de la solution présentée par rapport à un circuit de vérification basé sur la norme JTAG.
- Coût en surface (section 5.3.8) : définit l'impact du circuit de vérification sur la surface totale occupée par un système électronique. Ce paramètre permet aussi de savoir, dans le

cadre d'une implémentation sur FPGA, l'espace disponible pour l'ajout de fonctionnalités supplémentaires.

5.3.1 Mémoire

5.3.1.1 Analyse théorique du taux d'utilisation

Comme cela a été vu dans la section 4.5.1.2 sur le contrôleur, page 87, la gestion de la mémoire ne permet pas la fragmentation de l'information pour optimiser le taux d'utilisation. C'est-à-dire que si des données ne sont pas un multiple de 64, des bits « inutiles » sont inclus dans l'information plutôt que de prendre une partie de la prochaine donnée pour compléter le paquet.

On donne un graphique sur la Figure 5.8 représentant une analyse théorique du taux d'utilisation de la mémoire en fonction du nombre de canaux d'observations dans le module.

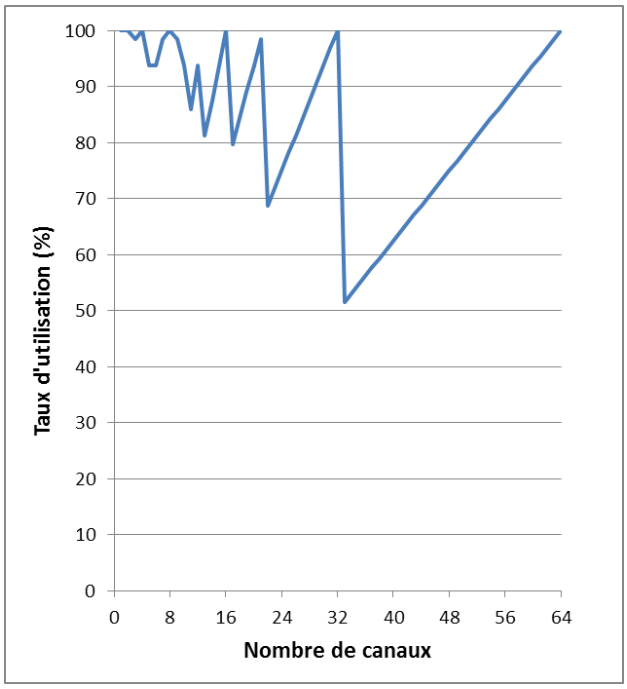


Figure 5.8 : Analyse théorique du taux d'utilisation de la mémoire en fonction du nombre de canaux d'observation.

Le nombre de canaux maximum est 64 selon les spécifications du projet. Le taux d'utilisation est toujours supérieur à 50 % et les maximums de 100 % correspondent aux puissances de 2. Afin d'avoir un taux d'utilisation de 100 % quel que soit le nombre de canaux, un bloc logique devrait être ajouté au contrôleur. Ce dernier aurait pour rôle de fragmenter les paquets de données qui

dépassent la largeur du bus de données afin de supprimer les bits inutiles. La Figure 5.9 donne un exemple de cette solution dans le cas où les paquets de données ont une taille de 20 bits. Dans ce cas, le 4ème paquet est fragmenté en deux partie, les 4 premiers bits lors du premier envoi sur le bus d'interface mémoire et les 16 bits suivants lors du deuxième envoi. Puis ce sera le 7ème paquet qui sera fragmenté, et ainsi de suite.

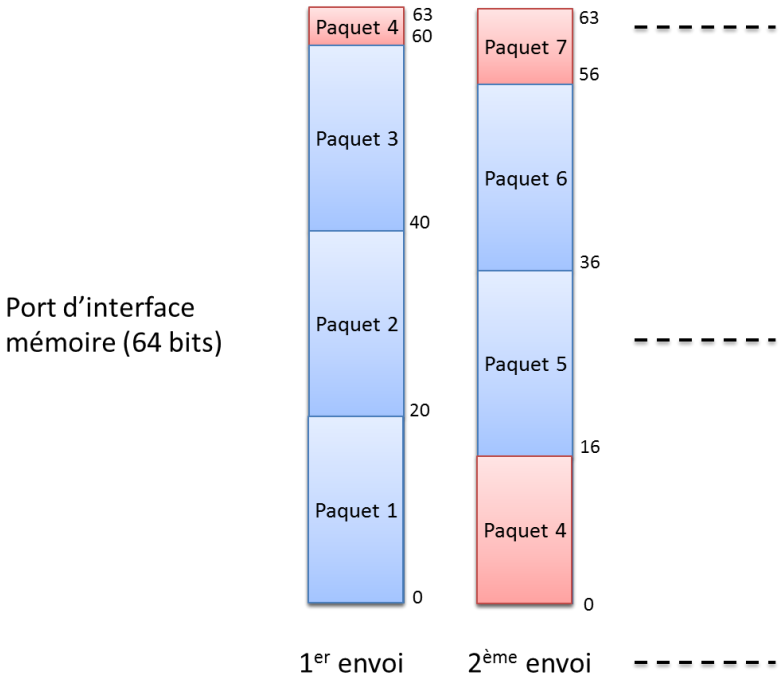


Figure 5.9 : Exemple de gestion des paquets (taille de 20 bits) de données envoyés en mémoire pour un taux d'utilisation de 100 %.

5.3.1.2 Bande passante

La bande passante théorique de la mémoire DDR2 utilisée est 2.5 Go/s :

Les mémoires de type DDR transfèrent des données sur les deux fronts de l'horloge. Elle possède un bus de 16 bits (2 octets) et une fréquence de 625 MHz : 625 MHz * 2 * 2 octets = 2.5 Go/s.

Le volume de données maximum créé par l'enveloppe P1500 est de 1.25 Go/s :

Le nombre de canaux d'observation (enregistrés en mémoire) est de 64 (8 octets de données) et la fréquence du système de 156.25 MHz : 156.25 MHz * 8 octets = 1.25 Go/s.

Il serait donc possible en théorie d'augmenter (ici jusqu'à 128) le nombre de canaux d'observation sans que la bande passante de la mémoire ne soit dépassée. Il serait aussi possible

d'avoir des cellules d'observation de l'enveloppe qui échantillonnent sur les deux fronts de l'horloge (doublant ainsi la bande passante nécessaire).

Pour augmenter le débit maximum, il faudrait augmenter la fréquence de la mémoire utilisé, ou interfacer des mémoires supplémentaires. Il serait aussi possible d'utiliser des mémoires internes du FPGA (appelées *Block RAM*). Cette solution permettrait de réduire la distance entre la production et le stockage des données, augmentant ainsi la vitesse maximale du débit.

5.3.2 Observation de signaux

L'observation de signaux dans le circuit sous test est compatible jusqu'à l'horloge du système, soit 156.25 MHz. L'acquisition des signaux est asynchrone puisqu'elle ne prend pas en compte une horloge externe pour une synchronisation. La fréquence d'acquisition doit alors être plusieurs fois plus grande que l'horloge du circuit sous test pour garantir une bonne qualité d'observation. Néanmoins, ce type d'acquisition ne permet pas d'identifier des impulsions dont la durée est courte devant la période d'échantillonnage.

La précision avec laquelle des signaux numériques peuvent être observés a été caractérisée en mesurant l'incertitude relative de mesure durant l'échantillonnage avec notre système. La Figure 5.10 donne un exemple de l'échantillonnage d'une impulsion.

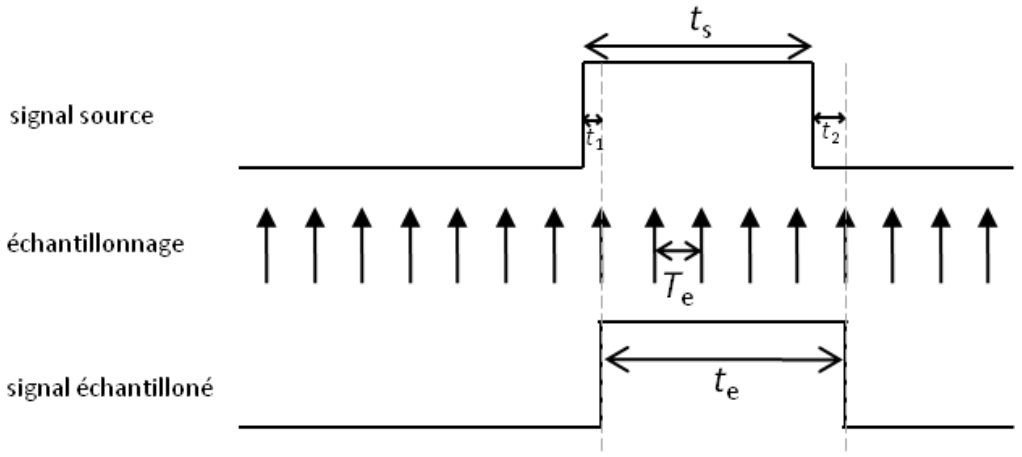


Figure 5.10 : Exemple d'échantillonnage d'une impulsion numérique – Le signal source est échantillonné à chaque période de durée T_e, donnant le signal échantillonné.

Il y a un signal source qui génère une impulsion d'une durée t_s qui est échantillonnée avec une période d'échantillonnage Te. Le signal échantillonné obtenu, t_e, est tel que :

$$t_s = t_e + (t_2 - t_1)$$

t_1 et t_2 sont les incertitudes de mesures dues au décalage entre l'instant de la transition du signal et l'instant d'échantillonnage. Ces incertitudes ont une valeur comprise entre 0 et $T_e/2$ donc l'incertitude de mesure totale est aussi comprise entre 0 et $T_e/2$. Ainsi, la valeur maximale de l'erreur d'une mesure est de $\pm T_e/2$. L'incertitude relative désigne dans le cas présent le rapport entre l'erreur maximale et la période du signal échantillonné. La méthode pour déterminer l'incertitude relative de mesure avec l'outil de vérification est alors décrite.

Un générateur de fonctions a été relié en entrée d'une cellule d'observation et les signaux échantillonnés, enregistrés puis lus en mémoire ont été analysés en fonction de la fréquence. Les signaux ont été analysés grâce à un script sur Matlab (en annexe : Figure A.IV et Figure A.V) qui permet de reconstruire le signal à partir des données échantillonnées et du signal « data_valid ». Ce script a également pour rôle d'analyser le signal reconstruit pour connaître les périodes des signaux. Ce sont les mesures de temps les plus courtes qui sont étudiées, c'est à dire à la mesure du temps où le signal est haut ainsi que le temps où le signal est bas. C'est donc sur ces mesures que l'incertitude relative de l'échantillonnage est donnée en fonction de la fréquence du signal en entrée (Figure 5.11).

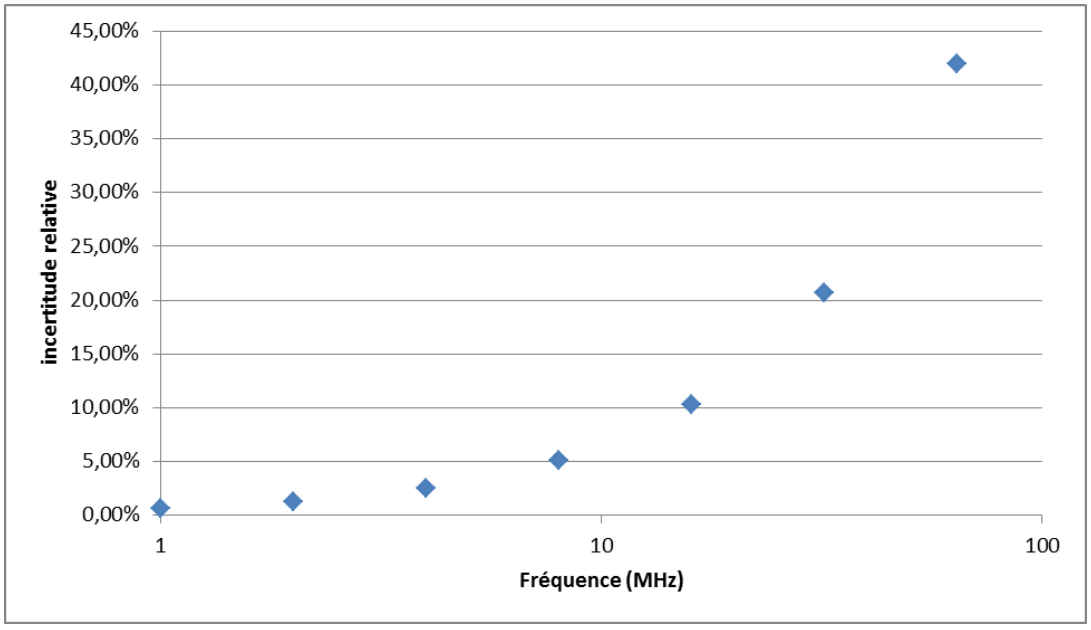


Figure 5.11 : Incertitude relative de l'échantillonnage en fonction de la fréquence du signal en entrée.

On remarque que lorsque l'on approche de la moitié de la fréquence d'échantillonnage (environ 78 MHz), limite théorique de la fréquence à laquelle il est possible d'échantillonner un signal, l'incertitude augmente de manière significative. La fréquence maximale à laquelle des signaux sont observables avec une bonne précision est d'environ 10 MHz (5 % d'incertitude relative). Si des impulsions ne dépassent pas cette fréquence, il est presque sûr qu'elles seront visibles après l'acquisition. Cependant, si le signal observé est répétable facilement, des fréquences plus élevées peuvent être atteintes puisque les résultats pourront ensuite être moyennés.

Concernant l'observation de signaux synchrones, il serait nécessaire de développer une enveloppe P1500 dont l'horloge provient du système sous vérification. La vitesse d'acquisition des signaux synchrone serait alors reliée à la limite de fréquence de commutation des entrées/sorties du FPGA, sans compter les délais de routage. Pour le Spartan 6, l'ordre de grandeur de la fréquence de commutation des entrées/sorties est de 400 MHz. Il est possible que la fréquence d'acquisition de signaux synchrone maximale soit comparable à celle pour les signaux asynchrones.

5.3.3 Contrôle de signaux

Le contrôle des signaux dans le circuit sous test est compatible jusqu'à l'horloge du système, soit 156.25 MHz. Cependant, ceci nécessite qu'un port d'entrée soit assez large pour fournir les données nécessaires à chaque coup d'horloge. Une solution future serait de créer de nouvelles instructions qui permettraient de charger des vecteurs de stimuli dans la mémoire DDR2 puis de le lire et de les appliquer sur les entrées désirées à 156.25 MHz ou plus.

5.3.4 Horloges

L'horloge principale du système est de 156.25 MHz et l'horloge de la mémoire est de 625 MHz.

Une fréquence maximale d'un circuit est obtenue quand la période de l'horloge est d'une durée suffisante et minimale pour que tous les signaux aient le temps de se propager. Il y a deux méthodes qui permettraient d'augmenter la fréquence du système : l'étude des chemins critiques et l'utilisation du FPGA sans la carte de développement.

D'abord, les chemins critiques représentent les expressions logiques, programmées dans le FPGA, qui sont les plus longues à traverser. Identifier ces dernières et les modifier permet

d'augmenter la fréquence du système tout en gardant les mêmes fonctionnalités. Une méthode connue pour compenser des chemins critiques est l'introduction de registres de pipelines. Ces derniers servent à couper un chemin en plusieurs morceaux, chacun devenant plus rapide à traverser. À la vue de l'analyse des temps, grâce au logiciel ISE, cette technique serait applicable à l'outil de vérification pour augmenter sa fréquence de fonctionnement.

Ensuite, si le FPGA mis en œuvre était utilisé dans le contexte du WaferBoard™, c'est-à-dire sans la carte de développement, les entrées/sorties du FPGA se retrouveraient à une plus faible distance des entrées/sorties des circuits sous test. Le temps de propagation des signaux entre les différentes entrées/sorties serait plus court et il serait ainsi possible d'augmenter la fréquence de l'outil.

Enfin, si la fréquence de fonctionnement de l'outil de vérification augmente, alors la quantité de données générées devient plus grande (voir la section 5.3.1.2). Il serait alors nécessaire d'augmenter la bande passante de la mémoire pour ne pas avoir un goulot d'étranglement dans le design, c'est-à-dire l'incapacité à gérer la totalité des données. D'après les spécifications des mémoires de type DDR2, leur fréquence de fonctionnement maximale est de 800 MHz. Néanmoins, des essais d'implémentation à cette fréquence ont résulté par des chemins critiques dans le MCB. Des expérimentations sont alors nécessaires pour vérifier si le module reste fonctionnel à 800 MHz.

5.3.5 Entrées/sorties

Les possibilités de l'outil de vérification sont directement liées au nombre d'entrées/sorties (E/S) disponibles sur le FPGA. Les besoins du design sont les suivants :

- Enveloppe P1500, 2 E/S par canal : entre 4 et 128 E/S (maximum de 64 canaux)
- DDR2 : 46 E/S
- Contrôle, dépend du nombre de canaux : entre 34 et 136 E/S

Au total, l'outil nécessite entre 84 et 310 E/S disponibles sur le FPGA cible en fonction du nombre de canaux de contrôle et d'observation désirés sur l'enveloppe P1500. La Figure 5.12 donne un graphique du nombre d'E/S nécessaires en fonction du nombre de canaux P1500.

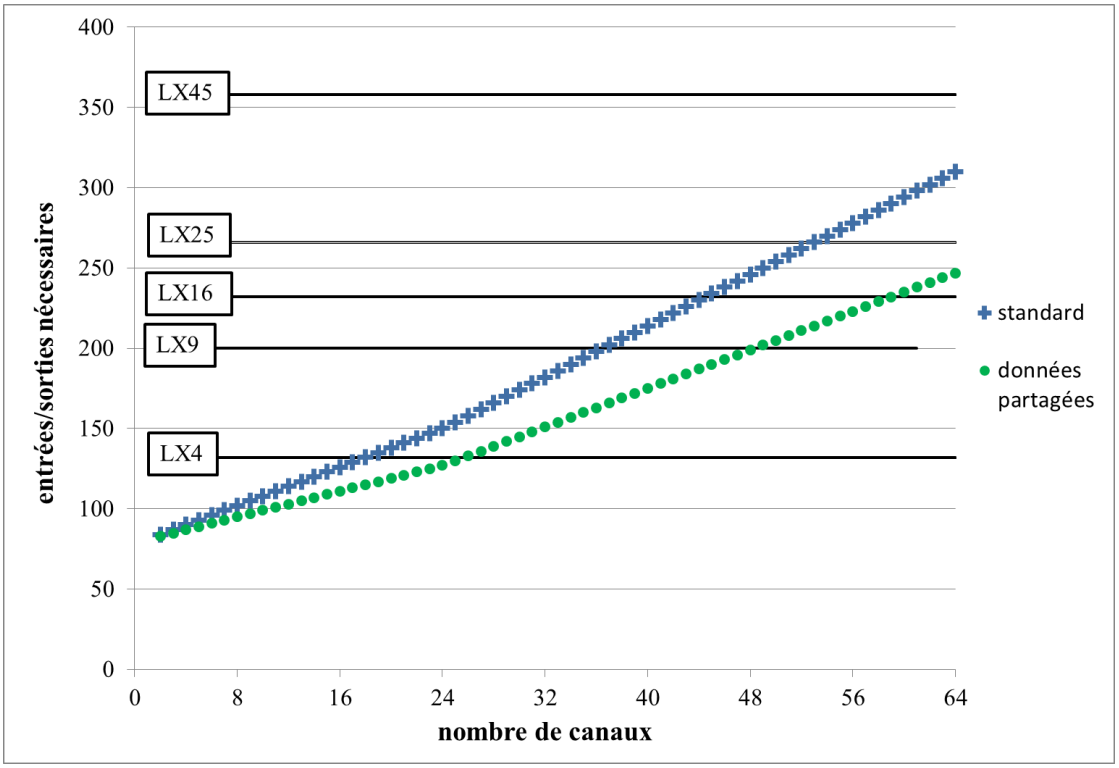


Figure 5.12 : Nombre d'E/S nécessaires pour réaliser l'outil sur FPGA en fonction du nombre de canaux P1500 désirés.

La courbe nommée « standard » correspond au circuit tel qu'il est aujourd'hui. La courbe nommée « données partagées » représente le cas où le bus de données qui fait l'interface du module serait partagé pour les données de vérification et les données de configuration. Des courbes horizontales ont été ajoutées et représentent le nombre d'E/S disponibles dans plusieurs distributions du FPGA Spartan6 de Xilinx de surface croissante, du LX4 le plus petit au LX45, c'est ce dernier qui est utilisé dans nos travaux. Selon ses besoins de vérification, un concepteur peut choisir des FPGA qui occupent plus ou moins de surface dans son système. Il existe des FPGA disposant d'encore plus d'E/S[8] et il est aussi possible de multiplier le nombre d'outils de vérification. De plus, il serait possible de distribuer l'enveloppe P1500 sur plusieurs FPGA de vérification pour augmenter le nombre d'entrées/sorties et donc de canaux maximum. Les FPGA seraient synchronisés et leur contrôle serait partagé.

[8] Le FPGA Virtex 7 XC7V2000T de Xilinx dispose de 1200 entrées/sorties

5.3.6 Analyseur logique

Les caractéristiques principales de l'analyseur logique réalisé dans ce projet sont les suivantes :

- Fréquence d'acquisition asynchrone : 156.25 MHz
- Nombre de canaux : 64
- Conditions de déclenchement : fronts ou niveau et association des conditions sur les différents canaux (jusqu'à 64 conditions)
- Durée d'échantillonnage maximale $T_{\max}$ à f=156.25 MHz et sur 64 canaux : 0.1 s. Le taux d'utilisation *tx* de la mémoire à 64 canaux est de 100 % donc on a :

$$T_{\max} = tx.\frac{N_{\text{mem}}}{f.N_{\text{canaux}}} = 1.\frac{10^9}{156.25 \times 10^6 \times 64} = 0.1s$$

  avec N_{mem} la taille de la mémoire en bits et N_{canaux} le nombre de canaux.

Les caractéristiques qui devraient être ajoutées dans l'analyseur logique sont :

- Un mode d'acquisition synchrone : cellules de l'enveloppe P1500 spécifiques et reliées à une horloge externe du système électronique sous vérification.
- Une fréquence d'échantillonnage configurable à l'aide d'un diviseur d'horloge.
- Des conditions de déclenchement plus complexes, comme la reconnaissance d'une séquence de bits, d'un protocole de communication ou d'assertions.

5.3.7 Réduction du temps de vérification

Un modèle simple est présenté ici, permettant d'estimer l'accélération du temps de vérification avec le système présenté à cellules parallèles en comparaison avec l'équivalent suivant la technologie JTAG sérielle. On considère une vérification dans laquelle il y a *N* canaux (cellules), *X* mises à jour (*upload* = vecteur de vérification) et *Y* captures de données.

Dans le système sériel, pour une mise à jour, *N* décalages (*shift*) sont nécessaires pour préparer le vecteur dans les *N* cellules puis faire l'opération mise à jour, on a en cycle d'horloge :

$$ts_{\text{upload}} = N(\text{shift}) + 1(\text{upload}) = N + 1$$

De même, pour une capture, l'opération capture puis *N* décalages permettent de récupérer les données, on a en cycles d'horloge :

$$ts_{\text{capture}} = 1(\text{capture}) + N(\text{shift}) = 1 + N$$

Ainsi, la durée de vérification sérielle est estimée à :

$$ts = X.ts_{\text{upload}} + Y.ts_{\text{capture}} = (N + 1)(X + Y)$$

Avec le système parallèle, il ne faut que 1 cycle pour chaque instruction, y compris le décalage (*shift*) dans *N* cellules en même temps.

Pour une mise à jour, on a en cycle d'horloge :

$$tp_{\text{upload}} = 1(\text{shift}) + 1(\text{upload}) = 2$$

Et pour une capture :

$$tp_{\text{capture}} = 1(\text{capture}) + 1(\text{shift}) = 2$$

La durée de vérification parallèle est donc de :

$$tp = X.tp_{\text{upload}} + Y.tp_{\text{capture}} = 2(X + Y)$$

Le rapport *R* de la durée de vérification sérielle sur la durée de la vérification parallèle est estimé selon l'expression suivante :

$$R = \frac{ts}{tp} = \frac{(N + 1)(X + Y)}{2(X + Y)} = \frac{N + 1}{2}$$

Le rapport *R* en fonction du nombre de canaux, correspondant au gain de temps de vérification avec l'outil présenté par rapport à un circuit JTAG, est représenté sur la Figure 5.13. Par exemple, pour 39 canaux utilisés, la vérification serait 40 fois plus rapide avec l'outil présenté dans ce mémoire. Le gain de temps est plus grand à mesure que l'on augmente le nombre de canaux. Cela est d'autant plus intéressant qu'un système électronique comporte de nombreux signaux auxquels il faut s'interfacer.

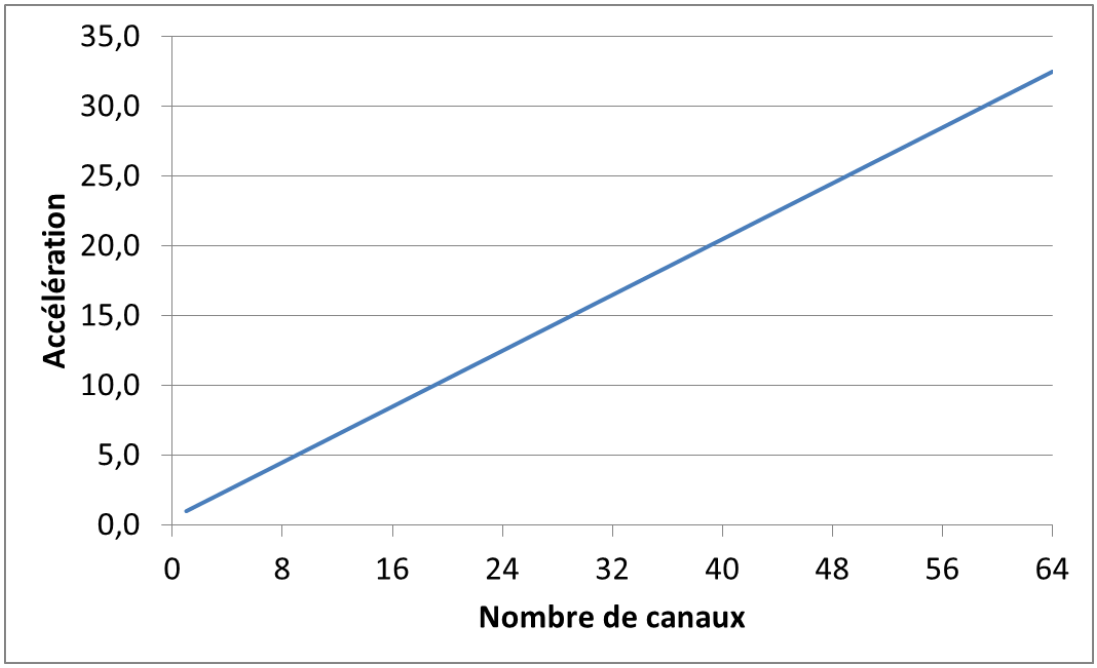


Figure 5.13 : Gain de temps de vérification avec l'outil présenté par rapport à un circuit de type JTAG en fonction du nombre de canaux.

5.3.8 Surface

La surface nécessaire dans le FPGA est relativement faible par rapport au total de registres et de LUTs (*Look-Up Table*) disponibles. La Figure 5.14 et la Figure 5.15 montrent la surface occupée en fonction du nombre de canaux. Ces résultats proviennent des rapports obtenus après une synthèse avec le logiciel ISE. Le nombre de canaux a un faible impact sur l'occupation en surface du FPGA. Il y a finalement beaucoup d'espace inutilisé dans le FPGA, laissant une opportunité de développer des modules qui complèteront les fonctionnalités actuelles de l'outil. Ce dernier pourrait avoir une enveloppe P1500 plus complexe avec des instructions spécifiques. Il pourrait également posséder un analyseur logique plus évolué. De plus, un module de communication avec l'extérieur n'a pas encore été développé.

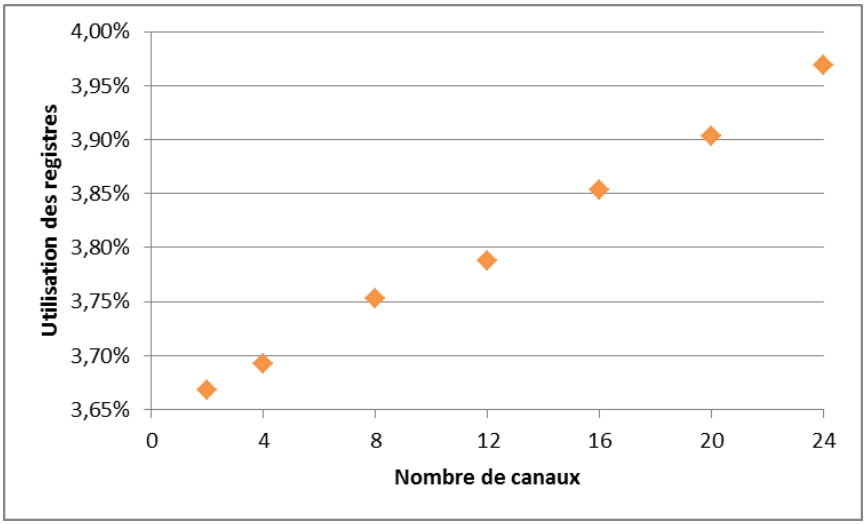


Figure 5.14 : Utilisation des registres du FPGA en fonction du nombre de canaux.

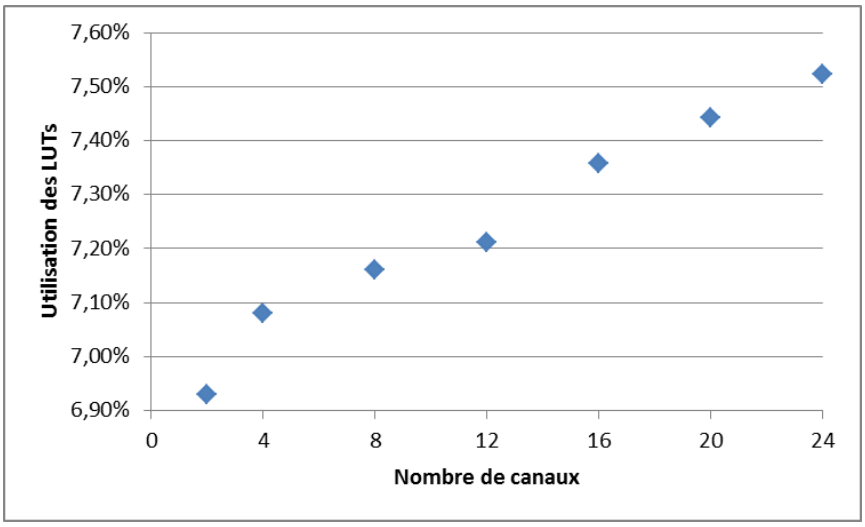


Figure 5.15 : Utilisation des LUTs du FPGA en fonction du nombre de canaux.

5.3.9 Bilan des caractéristiques et performances de l'outil

Au cours de ce chapitre, les caractéristiques et les performances de l'outil de vérification ont été exposées. De plus, des améliorations futures ont été proposées pour la plupart des points qui ont été abordés. Le Tableau 5.4 et le Tableau 5.5 forment un résumé du chapitre et offrent une vue globale des travaux présents et futurs de l'outil de vérification.

Tableau 5.4 : Bilan de caractéristiques et performances de l'outil de vérification – 1er de 2

Propriété	Caractéristiques/Performances	Améliorations futures
Taux d'utilisation de la mémoire.	Entre 50 et 100 % en fonction du nombre de canaux.	Taux à 100 % quel que soit le nombre de canaux grâce à la fragmentation des paquets de données.
Bande passante de la mémoire.	2.5 Go/s.	Utilisation de plusieurs mémoires. Utilisation de mémoire interne au FPGA.
Observation de signaux	Observation asynchrone : fréquence d'échantillonnage numérique à 156.25 MHz.	Observation synchrone : ajout de cellules spécifiques dans l'enveloppe P1500 reliées et synchronisées à une horloge externe (du système sous vérification).
Contrôle de signaux	Application de vecteurs à une fréquence de 156.25 MHz. Nombre de canaux simultanés limité à la largeur du bus d'entrée de l'outil de vérification.	Chargement des vecteurs de vérification dans la mémoire externe pour augmenter le nombre de canaux de contrôle simultanés.
Horloges	Système : 156.25 MHz Mémoire externe : 625 MHz	Augmentation de la fréquence du système en corrigeant les chemins critiques, notamment par l'introduction de registres de pipelines. Augmentation de la fréquence de la mémoire à 800 MHz, fréquence maximale, il serait alors nécessaire de vérifier les chemins critiques associés à cette augmentation.
Entrées et sorties	Entre 84 (pour 2 canaux) et 310 (pour 64 canaux) entrées/sorties sont nécessaires.	Le partage du bus de contrôle et de données en entrée de l'outil permettrait de limiter le nombre d'entrées/sorties maximales à 250 au lieu de 310. Distribution de l'enveloppe P1500 sur plusieurs FPGA de vérification pour augmenter le nombre de canaux maximum. Les FPGA seraient synchronisés et leur contrôle serait partagé.

Tableau 5.5 : Bilan de caractéristiques et performances de l'outil de vérification – 2ème de 2

Propriété	Caractéristiques/Performances	Améliorations futures
Analyseur logique	64 canaux avec une profondeur de 0.1 seconde par canal, à 156.25 MHz. Déclenchement configurable sur front ou niveau de plusieurs signaux.	Fréquence d'échantillonnage configurable grâce à un diviseur de fréquence. Conditions de déclenchement complexes : séquences de bits, protocoles de communication, assertions, …
Accélération de la vérification	Enveloppe P1500 permettant un accès parallèle, d'autant plus rapide que les circuits à accès sériel qu'il y a de canaux.	
Utilisation de la surface du FPGA	Registres : environ 4 % LUTs environ 7 %	

CHAPITRE 6 CONCLUSION

6.1 Synthèse des travaux

Le projet DreamWafer™ a pour objectif de produire le WaferBoard™, un outil de prototypage rapide de systèmes électroniques. Ce dernier propose une surface composée d'une matrice de contacts reliés à un réseau dont les interconnexions sont reconfigurables. Il est destiné à recevoir et interconnecter un ensemble de circuits intégrés afin de construire un système électronique. La preuve de concept du WaferBoard™ est en bonne voie pour réussir. Un banc de test dédié a été conçu et malgré plusieurs incertitudes les tests électriques se sont avérés positifs. Il est aussi apparu que la carte d'alimentation et de configuration du MiniWaferIC, le PowerBlock™ fonctionne de la manière attendue. Enfin, les vérifications effectuées sur le MiniWaferIC ont démontré que le réseau de configuration JTAG était fonctionnel.

Pour pouvoir utiliser le WaferBoard™ et prototyper un système électronique, il est nécessaire que des circuits soient imaginés pour que la vérification et le débogage puissent être accomplis. Au cours de mes activités de recherches, la littérature a d'abord été parcourue pour cerner les enjeux et les tendances du domaine de la vérification et du débogage des systèmes électroniques. La conclusion est qu'il existe de multitude méthodes reconnues pour effectuer la vérification et parvenir à la validation des circuits. Certains optent pour l'émulation matérielle grâce aux FPGA ou au développement sur des émulateurs matériels de performance, ils s'exposent alors au risque de ne pas avoir accès au modèle matériel de chaque module du système électronique à vérifier et à valider. D'autres préfèrent insérer des modules dédiés au sein de leurs designs. Ces derniers ont un coût en surface non négligeable, mais permettent d'accélérer de manière significative la vitesse de la vérification, dont la part ne cesse de grandir dans la chaîne de conception. En outre, ils permettent la vérification sur les circuits intégrés réels et non pas sur des modèles matériels. De plus, le WaferBoard est aussi une plateforme qui permet la vérification de systèmes électroniques sans connaître les modèles de tous les circuits intégrés qui le composent.

L'idée retenue pour la vérification d'un système électronique avec le WaferBoard™ est de l'instrumenter avec un outil de vérification in-situ sur FPGA. Cet outil rassemble des fonctionnalités pour pouvoir effectuer la vérification et le débogage d'un système électronique. Il possède un circuit permettant de s'interfacer avec des circuits intégrés, un analyseur logique

embarqué configurable et un décodeur d'instructions. Cet outil doit être placé in-situ, parmi les autres circuits intégrés du système électronique sous vérification, ainsi au plus près des signaux qui doivent être vérifiés. Cette proximité permet d'obtenir des fréquences de fonctionnement plus grandes et de meilleures mesures de délais. La norme IEEE 1500 a été choisie pour le circuit d'interface :

- Héritière du JTAG, elle a été développée pour faciliter les échanges de circuits entre concepteurs (acheteurs et vendeurs) en proposant de normaliser les interfaces pour les circuits intégrés.
    - Enveloppe de cellules à balayage (*scan*) d'observation et de contrôle
    - Architecture dédiée au test
- Supportée par un langage de description matérielle associé, le CTL, pour lequel des outils de synthèse, de génération automatique et de vérification de la conformité existent.
- De nombreuses études ont été publiées sur des méthodologies efficaces ou pour répondre à des besoins spécifiques de la vérification et du débogage avec l'IEEE 1500, comme la présence de plusieurs domaines d'horloges par exemple.

Enfin, une première version de l'outil de vérification reconfigurable a été implémentée sur une carte de développement FPGA Xilinx Spartan 6, en vue de faire une preuve de concept. Le résultat de ce travail est un module compatible IEEE 1500 sur FPGA qui permet à un concepteur d'encapsuler un ou plusieurs circuits dans une enveloppe et de procéder à la vérification et au débogage du système électronique sous vérification. L'outil peut s'interfacer avec jusqu'à 64 entrées ou sorties à travers des cellules IEEE 1500. Ces cellules permettent l'observation ou le contrôle des signaux, avec un accès en parallèle, afin de tester les fonctionnalités internes ou externes d'un circuit. Le design contient un analyseur logique configurable qui permet de détecter des conditions de déclenchement simples et d'enregistrer des signaux numériques dans une mémoire externe DDR2. Les mémoires DDR2 procurent un espace de stockage important et une vitesse de communication rapide pour répondre à l'importante quantité de données qui sont générées par l'enregistrement de nombreux signaux numériques. L'outil contient également un décodeur d'instructions qui permet de contrôler le cœur P1500 et de communiquer avec l'analyseur logique et la mémoire. Les fonctionnalités du système ont été vérifiées, notamment à

travers le contrôle et l'observation d'une communication SPI en provenance d'un microcontrôleur PIC.

6.2 Avantages et limitations du système proposé

L'outil de vérification proposé dans ce mémoire est une solution pertinente pour la vérification des systèmes électroniques puisqu'il permet l'observation et le contrôle de signaux. Il peut être caractérisé par plusieurs avantages spécifiques :

- Implémentation reconfigurable sur FPGA : permet de reconfigurer dynamiquement l'outil de vérification en fonction des besoins de l'utilisateur.
- Le circuit d'interface basé sur une norme, l'IEEE 1500 : permet une meilleure portabilité et compréhension de l'outil dans différents projets ou systèmes électroniques.
- Accès en parallèle aux signaux de l'enveloppe IEEE 1500 pour réduire la longueur des chaînes de balayage à une unique cellule. Ainsi, l'outil peut exécuter une instruction par cycle quand il est configuré en mode parallèle.
- Configuration in-situ de l'outil de vérification : permet de modifier les points d'accès aux interfaces d'un système électronique sans avoir à déplacer physiquement des sondes.
- Utilisation d'une mémoire externe rapide : permet d'avoir un meilleur espace de stockage pour les données observées sans limiter la bande passante du système.
- Solution directement utilisable sur le WaferIC™ comme outil de vérification et de débogage spécifique pour le WaferBoard™. Elle peut aussi être utilisée sur un PCB, dans un émulateur matériel ou dans un FPGA faisant partie d'un système électronique. Cette dernière technique est connue pour économiser de l'espace dans un système. Un FPGA accueille des circuits DFT pendant la phase de vérification et il peut ensuite s'occuper d'autres tâches lors du fonctionnement réel.

Cependant, plusieurs points pourraient être améliorés, constituant les limites de l'outil présenté :

- La fréquence de fonctionnement est limitée, actuellement à 156.25 MHz, car l'outil est embarqué sur un FPGA. Cette grandeur est petite devant les fréquences en gigahertz des analyseurs logiques externes, mais est similaire à d'autres solutions embarquées comme

Chipscope. Une fréquence trop faible empêche de réaliser une vérification de qualité, car elle ne permet pas d'observer toutes les transitions sur les signaux. Il est néanmoins possible de considérer la solution proposée sur un FPGA à gravure plus fine qui permettrait d'augmenter la fréquence.

- L'outil de vérification est limité par le nombre d'entrées/sorties du FPGA qui accueille le design. Dans la version actuelle, le FPGA permettrait d'avoir jusqu'à 101 canaux et l'implémentation actuelle définit une limite de 64 canaux. Si un concepteur a besoin d'un grand nombre de canaux pour la vérification de son système, il devra considérer un FPGA avec beaucoup d'entrées/sorties ou l'utilisation de plusieurs outils en même temps.
- La quantité de mémoire actuelle correspond à une durée d'enregistrement des signaux de 0.1 seconde à 156.25 MHz et 64 canaux. Or si seule la mémoire interne était utilisée, comme avec Chipscope, cette durée serait limitée à 40 µs.

6.3 Améliorations futures

L'implémentation faite dans le cadre de mes travaux de recherche se voulait preuve de concept. Cette section dresse plusieurs améliorations possibles pour en améliorer les fonctionnalités ou les performances.

6.3.1 Multiplication des FPGA de vérification

Pour pallier le manque d'entrées/sorties du FPGA de vérification, une solution intéressante serait de multiplier le nombre de FPGA utilisés (Figure 6.1).

Plusieurs FPGA partageraient leurs entrées/sorties pour augmenter le nombre de cellules P1500 disponibles pour la vérification et le débogage du système sous test. Un circuit maître serait dédié au contrôle et à la synchronisation des différents FPGA de vérification. La multiplication des FPGA de vérification pourrait ainsi permettre de placer chaque outil au plus près du circuit sous vérification qui lui correspond pour réduire les délais de propagation et maximiser les fréquences d'injection et d'observation.

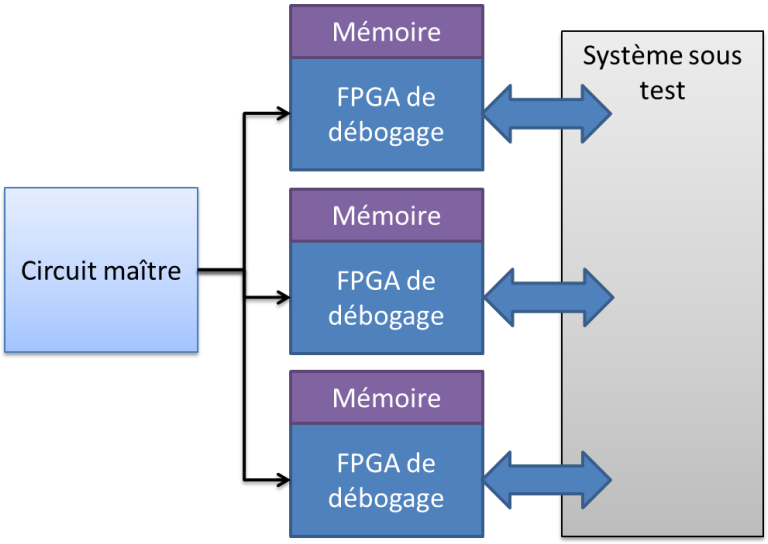


Figure 6.1 : Schéma de principe de la multiplication des FPGA de vérification

6.3.2 Interface utilisateur

Il n'existe actuellement aucun lien entre un logiciel d'acquisition et l'outil de vérification proposé. Le lien qui a permis de valider les fonctionnalités du module était Chipscope Pro. Il conviendrait alors de développer une interface utilisateur qui permettrait de configurer le FPGA selon les besoins du concepteur. De plus, un circuit de communication entre l'outil et l'ordinateur doit être élaboré. Une communication lente, comme l'UART, limite le nombre d'entrées/sorties et une communication rapide, telle que celles offertes par les interfaces SERDES et PCIe, permettrait d'augmenter la profondeur d'acquisition, la vitesse de la vérification et la vitesse de configuration. Le port de communication est aussi un module qui pourrait être reconfigurable pour s'adapter aux besoins de l'utilisateur.

6.3.3 Intégration à la plateforme de prototypage WaferBoard ™

L'outil de vérification a été imaginé pour fonctionner en complément du WaferBoard™. Les utilisateurs déposeraient leurs composants sur le WaferIC™ avec l'outil et une mémoire à proximité qui leur donneraient la visibilité nécessaire à la vérification et au débogage de leur système (Figure 6.2).

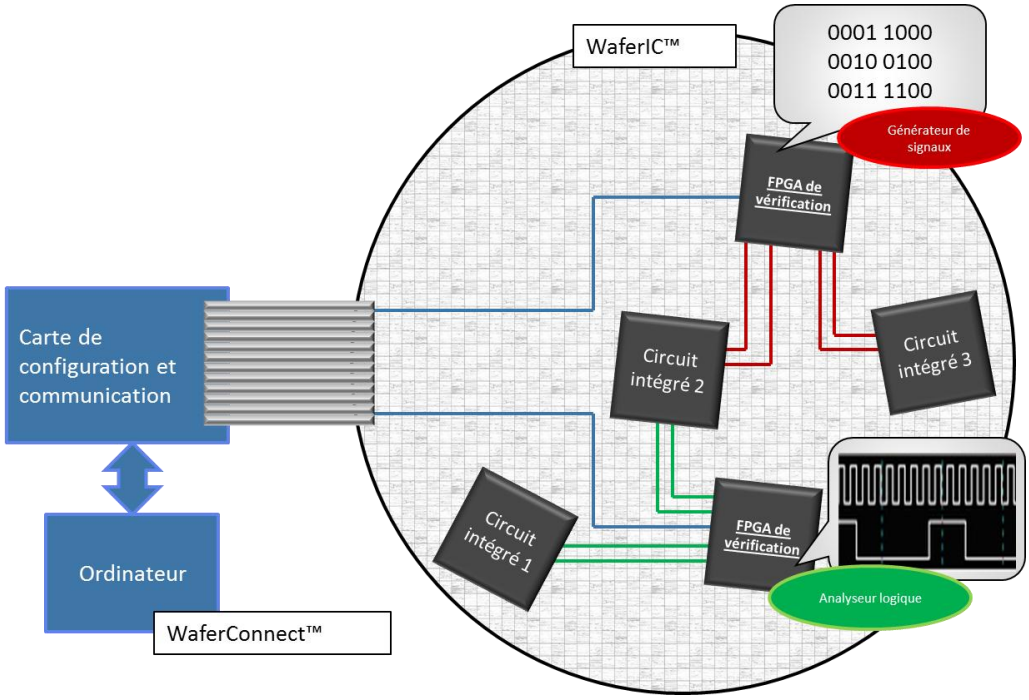


Figure 6.2 : Schéma de principe : utilisation de l'outil de vérification et de débogage dans le projet DreamWafer™

Cette utilisation de l'outil de vérification nécessiterait cependant de nouveaux développements :

- Implémentation d'un moyen de communication dédié avec la carte de configuration et de communication
- Caractérisation et compensation des délais de routage entre les outils de vérification et les circuits intégrés sous test. L'influence de la manière dont sont placés les outils sur le WaferIC™ pourrait aussi être caractérisée, comme la fréquence maximale en fonction de la distance avec les circuits sous test.
- Ajout d'une fonctionnalité spécifique à la vérification dans le logiciel WaferConnect™.

Advenant un manque d'espace sur le WaferIC™, seuls des FPGA présents sur la carte de configuration et communication pourraient être utilisés. Les limitations en fréquence de fonctionnement seraient alors plus fortes, car la longueur de la liaison avec les circuits sous test serait plus grande. De plus, l'outil de vérification aurait un nombre de canaux limité par la taille du lien entre le WaferIC™ et du reste du WaferBoard™, bien inférieure au nombre d'entrées/sorties disponibles sur un ou plusieurs FPGA embarqués.

RÉFÉRENCES

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, “Design for Testability,” in in *Digital Systems Testing and Testable Design*, 1990, pp. 343–419.

[2] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, D. Miller, S. Street, and S. Floor, “A Reconfigurable Design-for-Debug Infrastructure for SoCs,” pp. 7–12, 2006.

[3] S. Engineering, S. Committee, C. S. Committee, and C. Society, *IEEE Standard for System and Software Verification and Validation IEEE Computer Society*, vol. 2012, no. May. 2012.

[4] R. Norman, E. Lepercq, Y. Blaquière, O. Valorge, Y. Basile-bellavance, R. Prytula, Y. Savaria, and É. P. De Montréal, “An Interconnection Network For A Novel Reconfigurable Circuit Board,” no. 1, pp. 4–7, 2008.

[5] R. Norman, O. Valorge, Y. Blaquière, E. Lepercq, Y. Basile-bellavance, Y. El-alaoui, R. Prytula, Y. Savaria, and É. P. De Montréal, “An Active Reconfigurable Circuit Board,” vol. 1, no. 1, pp. 3–6, 2008.

[6] Intel, “Intel® Pentium® 4 Processor 1.50 GHz, 256K Cache, 400 MHz FSB - Specifications.” [Online]. Available: http://ark.intel.com/Products/Spec/sl4sh.

[7] accellera, *System Verilog 3.1a Language Reference Manual*. 2004.

[8] IEEE Computer Society, *IEEE Standard for the Fuctionnal Verification Language e*, vol. 2011, no. August. 2011.

[9] Accelera, “Universal Verification Methodology (UVM) 1.1 Class Reference,” no. June, 2011.

[10] Cadence Design Systems and Mentor Graphics, “OVM Class Reference,” 2011.

[11] accellera, *Property Specification Language Reference Manual*. 2004.

[12] K. K. C. Chen, “Tutorial (T-4) Assertion-Based Verification For SoC Designs,” pp. 12–15, 2003.

[13] K. Oh, S. Yoon, and S. Chae, “Emulator environment based on an FPGA prototyping board,” in *Proceedings 11th International Workshop on Rapid System Prototyping. RSP 2000. Shortening the Path from Specification to Prototype (Cat. No.PR00668)*, 2000, pp. 72–77.

[14] Xilinx, “UG029 Chipscope Pro Software and Cores User Guide,” 2013.

[15] Altera, “13. Design Debugging Using the SignalTap II Logic Analyzer,” in in *Quartus II Handbook Version 13.0*, vol. 3, no. May, 2013.

[16] accellera, *Open Verification Library*. 2002.

[17] Mentor Graphics, “A CLOSER LOOK AT VELOCE TECHNOLOGY : TAKING HARDWARE-ASSISTED VERIFICATION TO THE NEXT LEVEL,” no. May, 2009.

[18] Mentor Graphics, “ESL Simulation with Veloce Hardware Emulation,” 2012. [Online]. Available: http://www.mentor.com/esl/multimedia/esl-simulation-veloce-hardware-emulation-webinar.

[19] Nvidia, “SNEAK PEEK: INSIDE NVIDIA’S EMULATION LAB,” 2011. [Online]. Available: http://blogs.nvidia.com/blog/2011/05/16/sneak-peak-inside-nvidia-emulation-lab/.

[20] J. Hogan, “Hogan compares Palladium, Veloce, EVE ZeBu, Aldec, Bluespec, Dini,” 2013. [Online]. Available: http://www.deepchip.com/items/0522-04.html.

[21] B. Vermeulen, C. Hora, B. Kruseman, E. Jan, and M. Robert, “Trends in Testing Integrated Circuits,” pp. 688–697, 2004.

[22] IEEE Computer Society, *IEEE Std 1149.7-2009, IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture*, no. February. 2010.

[23] B. Quinton and S. Wilton, “Programmable logic core based post-silicon debug for SoCs,” *4th IEEE Silicon Debug Diagnosis Work.*, 2007.

[24] IEEE Computer Society, *IEEE Std 1500^{TM}-2005 IEEE Standard Testability Method for Embedded Core-based Integrated Circuits*, no. August. 2005.

[25] K.-J. Lee, T.-Y. Hsieh, C.-Y. Chang, Y.-T. Hong, and W.-C. Huang, “On-Chip SOC Test Platform Design Based on IEEE 1500 Standard,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 18, no. 7, pp. 1134–1139, Jul. 2010.

[26] L. Wang, R. Apte, S. Wu, B. Sheu, K. Lee, X. Wen, W. Jone, C. Yeh, W. Wang, H. Chao, J. Guo, J. Liu, Y. Niu, Y. Sung, C. Wang, F. Li, S. Technologies, and S. P. Ave, “Turbo1500 : Toward Core-Based Design for Test and Diagnosis Using the IEEE 1500 Standard,” pp. 1–9, 2008.

[27] F. Da Silva, T. McLaurin, and T. Waayers, *The Core Test Wrapper Handbook*, vol. 35. Springer US, 2006, p. 276.

[28] K. Petersén, “IEEE P1500 , Boundary Scan for SoCs,” vol. 1, no. 4, pp. 4–7.

[29] F. Dasilva, Y. Zorian, L. Whetsel, K. Arabi, R. Kapur, P. M. C. S. Burnaby, and B. C. C. Karimarabipmc-sierracom, “Overview of the IEEE P1500 Standard,” 2003.

[30] K. Chakravadhanula and V. Chickermane, “Automating IEEE 1500 Core Test - An EDA Perspective,” pp. 6–15, 2009.

[31] W. Huang, C. Chang, and K. Lee, “Toward Automatic Synthesis of SOC Test Platforms,” pp. 1–4, 2007.

[32] IEEE Computer Society, *IEEE Standard Test Interface Language (STIL) for Digital Test Vector Data — Core Test Language*, no. April. 2006.

[33] R. Kapur, B. Keller, M. Lousberg, P. Reuter, T. Taylor, and D. Kay, “CTL the Language for Describing Core-Based Test,” 2001.

[34] A. Benso, S. Member, S. Di Carlo, and P. Prinetto, “IEEE Standard 1500 Compliance Verification for Embedded Cores,” vol. 16, no. 4, pp. 397–407, 2008.

[35] E. J. Marinissen and Y. Zorian, “IEEE Std 1500 Enables Modular SoC Testing,” *IEEE Des. Test Comput.*, vol. 26, no. 1, pp. 8–17, Jan. 2009.

[36] E. J. Marinissen, “Analysis of The Test Data Volume Reduction Bene fi t of Modular SOC Testing Ozgur Sinanoglu,” 2008.

[37] H. Yi, S. Kundu, S. Cho, and S. Park, “A Scan Cell Design for Scan-Based Debugging of an SoC With Multiple Clock Domains,” *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 57, no. 7, pp. 561–565, Jul. 2010.

[38] P.-L. Chen, Y.-C. Huang, and T.-Y. Chang, “Fast Test Integration: Toward Plug-and-Play at-Speed Testing of Multiple Clock Domains Based on IEEE Standard 1500,” vol. 29, no. 11, pp. 1837–1842, 2010.

[39] P. Chen, J. Lin, and T. Chang, “IEEE Standard 1500 Compatible Delay Test Framework,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 17, no. 8, pp. 1152–1156, Aug. 2009.

[40] Y. Huang and J. Li, “Low-Cost Self-Test Techniques for Small RAMs in SOCs,” vol. 20, no. 11, pp. 2123–2127, 2012.

[41] A. Das, Ü. Kocaba, A. Sadeghi, I. Verbauwhede, K. U. Leuven, and E. Cosic, “PUF-based Secure Test Wrapper Design for Cryptographic SoC Testing,” 2012.

[42] M. Poppitz, “sump.org: FPGA Based Logic Analyzer,” 2007. [Online]. Available: http://www.sump.org/projects/analyzer/. [Accessed: 17-Jan-2013].

[43] Xilinx, “UG388 - Spartan-6 FPGA Memory Controller User Guide,” 2010.

[44] B. Vertenten, B. Okur, and B. Anderson, “UTMI+ Low Pin Interface (ULPI) Specification,” 2004.

BIBLIOGRAPHIE

Cartes de développement

Xilinx (2010) *Spartan 6 FPGA Memory Controller*. User's Guide 388.

Digilent (2012) *Atlys Board Reference Manual*.

Microchip (2005) *Explorer 16 Development Board User's Guide*.

Microchip (2012) PIC24HJXXXGPX06A/X08A/X10A Datasheet.

Xilinx (2013) *Spartan-6 FPGA SelectIO Resources*. User's Guide 381.

FPGA et émulateurs matériels

Auspy (avril 2012) Auspy [En ligne]. disponible : http://www.auspy.com/

BEEcube (2012) BEEcube [En ligne]. disponible : http://beecube.com/

Cadence Palladium (avril 2012) Cadence Palladium [En ligne]. disponible : http://www.cadence.com/products/sd/palladium_series/pages/default.aspx

EVE Zebu-Server (avril 2012) EVE Zebu-Server [En ligne]. disponible : http://www.eve-team.com/products/server_asic_emulator.html

Mentor Graphics Veloce (avril 2012) Mentor Graphics Veloce [En ligne]. disponible : http://www.mentor.com/products/fv/emulation-systems/

Group DINI (avril 2012) Group DINI [En ligne]. disponible : http://www.dinigroup.com/new/index.php

Jeff Ruedinger, IBM (2001) *Corp Knowledge of clocking aids ASIC-emulator choice* [En ligne] EDN. disponible : http://www.edn.com/article/491445-Knowledge_of_clocking_aids_ASIC_emulator_choice.php

Juergen Jaeger, Synopsys Inc. *FPGA-based rapid prototyping of ASIC, ASSP, and SoC designs* [En ligne] EE Times. disponible : http://www.eetimes.com/design/programmable-logic/4015246/FPGA-based-rapid-prototyping-of-ASIC-ASSP-and-SoC-designs?pageNumber=1

Moretti Gabe (novembre 2005) *Hardware Emulators Evolve to Meet ESL Demands* [En ligne] Chip Design. disponible : http://chipdesignmag.com/display.php?articleId=304&issueId=13.

Synopsys HAPS (2012) Synopsys HAPS [En ligne]. disponible : http://www.synopsys.com/systems/fpgabasedprototyping/pages/haps.aspx.

Veridae (avril 2012) Veridae [En ligne]. disponible : http://www.veridae.com/.

ANNEXES

A. Développements pour le projet DreamWafer™

Plusieurs tâches de développement dans le cadre du projet DreamWafer™ ont été abordées au cours de cette maîtrise. Faisant face à une importante courbe d'apprentissage, elles ont permis une meilleure compréhension des différents modules du WaferBoard™. De plus, elles ont contribué aux avancées techniques dont le projet a besoin pour parvenir à la construction finale du WaferBoard™. Dans un premier temps, le protocole de communication entre l'ordinateur et les différents modules du WaferBoard est résumé. Dans un deuxième temps, un logiciel pour le microcontrôleur sur le BottomPCB, pour la gestion de capteurs, de ventilateurs et de cartes auxiliaires, est décrit.

Protocoles de communication

De nombreux ports de communication, répondants aux spécifications des différents modules, sont utilisés dans le WaferBoard™. La Figure A.I donne un schéma des différents canaux de communication. Pour pouvoir communiquer avec le WaferIC™, le logiciel WaferConnect™ construit des paquets USB qui sont transmis à un FPGA, Spartan6 LX45 de Xilinx, dédié à la redirection des communications et placé sur un PCB, nommé Bottom PCB. Les paquets USB incluent un en-tête qui leur permet d'être redirigés au bon destinataire. Parmi les destinataires, il y a 21×4 bus de communication JTAG vers différentes zones du WaferIC™ ainsi qu'une liaison SPI avec un microcontrôleur PIC24H de Microchip. De plus, le microcontrôleur lui-même doit gérer un protocole I2C pour contrôler des cartes auxiliaires et les rails de ventilateurs qui servent à refroidir le WaferBoard™.

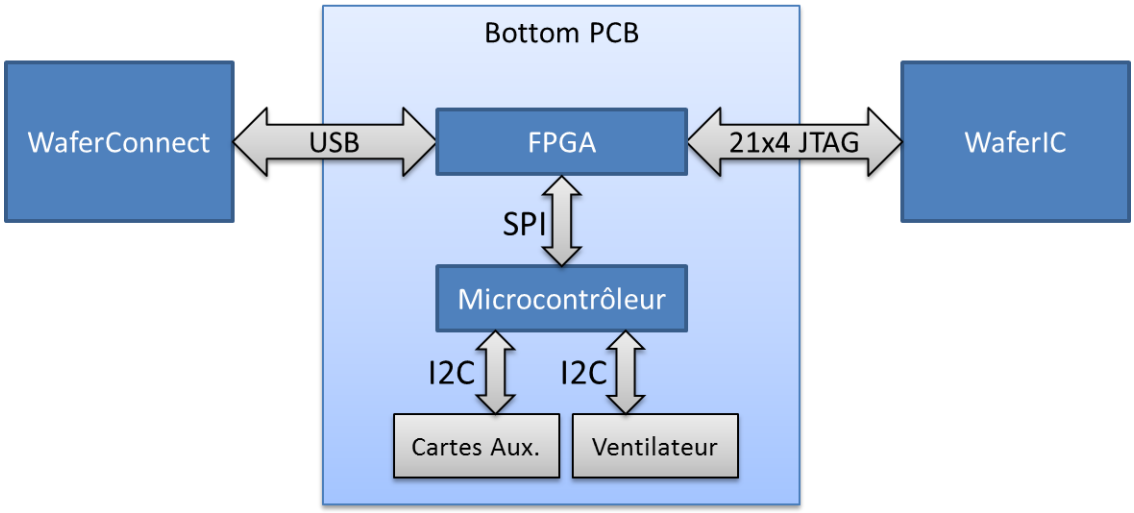


Figure A.I : Schéma des principaux canaux de communications dans le WaferBoard™.

Tous ces bus de communications doivent être utilisés selon des spécifications bien précises qui dépendent du fonctionnement de chaque composant, mais aussi des fonctionnalités qui y ont été implémentées. Cependant, les spécifications concernant la communication USB n'étaient pas à jour vis-à-vis des travaux courants et la communication SPI n'était pas définie. J'ai repris les documents de spécifications pour dresser un unique document concernant l'ensemble des protocoles de communication dans le Bottom PCB. En collaboration avec plusieurs stagiaires, l'architecture des protocoles a été revue pour être compatible avec les fonctionnalités voulues. Elle a aussi été adaptée pour être compatible avec les spécificités des différents modules qui la composent, notamment les interfaces du FPGA avec le logiciel d'un côté et avec le microcontrôleur de l'autre. Ce travail est avant tout la conception et l'optimisation d'un protocole de communication pour une architecture système fixée.

Le document produit[9] donne l'ensemble des informations nécessaires pour la programmation des interfaces entre le logiciel et le matériel. D'abord, il décrit les caractéristiques des interfaces physiques de communication présentes entre les composants du Bottom PCB. Par exemple, entre l'ordinateur de contrôle et le FPGA, il y a une interface USB2.0 du côté de l'ordinateur et ULPI [44] (UTMI+ Low Pin Interface) du côté du FPGA, un circuit USB3300 faisant un pont entre les deux. La norme ULPI permet d'interfacer un design avec un circuit intégré de

[9] Le document est nommé « Bottom PCB - Protocoles de communication »

transmission USB. Ensuite, le document détaille la structure et la signification des trames de données qui circulent entre les composants. Un exemple est donné sur la Figure A.II et décompose la trame en champs associés à une taille en bits.

Champ	Flag	Source	Destination	Subdestination	Frame ID	Length	Data	CRC
Taille	8 bits	2 bits	2 bits	5 bits	18 bits	13 bits	0 to 507 bytes	8 bits

Figure A.II : Extrait du document « Bottom PCB — Protocoles de communication » — Représentation des différents champs d'une trame entre le PC et le FPGA.

Enfin, il y a des listes d'instructions, adaptées au point de vue du logiciel WaferConnect™, qui permettent de mettre en œuvre des fonctionnalités du système. Un exemple est donné sur la Figure A.III. Il correspond à l'écriture du registre d'état du FPGA. Les différents champs, vus sur la Figure A.II, sont détaillés pour la commande présentée. Il y a aussi une représentation du registre de destination.

Écriture du registre d'état du FPGA BPCB

Cette commande permet de définir la valeur du registre d'état du FPGA BPCB

Commande

- **Flag** 1010 0001 (unique)
- **Source** 00 (PC)
- **Destination** 10 (FPGA BPCB)
- **Subdestination** 00101 (Registre d'état)
- **Length** 00 0000 0001 111 (= 1 octet entier)
- **Data** 1 octet

REL_EN	FPGA_MODE (00)	PIC_BOOTUP_MODE (000)	DONE	OVER_ CURRENT
bit 7				bit 0

Figure A.III : Extrait du document « Bottom PCB — Protocoles de communication » — Définition d'une instruction (partielle).

Ce document est finalement devenu une référence pour le développement de la couche physique du logiciel WaferConnect™ et pour les implémentations sur le FPGA et sur le microcontrôleur du Bottom PCB.

Développement sur le microcontrôleur

Le microcontrôleur du Bottom PCB est relié à des cartes auxiliaires et des ventilateurs avec plusieurs bus I2C. Ces canaux de communication nécessitent un logiciel qui permet d'optimiser les performances du système. De plus, le microcontrôleur doit être implémenté pour permettre la lecture de la mesure d'un capteur de courant. Cette mesure est une valeur analogique et doit être convertie en données numériques. Elle ne peut être réalisée dans le FPGA.

En lien avec le document sur les protocoles de communication décrit dans la partie précédente, une première version de l'implémentation du microcontrôleur PIC24H a été proposée. Cette tâche a été réalisée en collaboration avec une stagiaire[10]. Les fonctionnalités suivantes ont été validées sur la carte de développement Explorer 16 de Microchip pour PIC24H :

- Lecture des mesures du capteur de courant : les mesures de courant s'inscrivent dans la procédure du « *boot up* » (la mise en route) du Bottom PCB. Le but de ces mesures est de vérifier s'il n'y a pas de courts-circuits dans les différentes cellules du WaferIC™. Cette procédure correspond à un mode particulier du PIC, contrôlé par le FPGA du Bottom PCB, dans lequel toutes les autres fonctionnalités sont mises en attente. Le principe consiste à plusieurs mesures de courant successives qui sont comparées à des seuils connus par le PIC. À chaque mesure de courant, seule une petite partie du WaferIC™ est activée (4 réticules) par le FPGA pour identifier les régions fonctionnelles ou défectueuses. Les fonctionnalités de mesure et de comparaison avec un seuil du courant ont été validées grâce à un potentiomètre sur la carte de développement. Dans les autres modes du PIC, comme le mode fonctionnel, le capteur de courant est ignoré.
- Communication SPI avec le FPGA du Bottom PCB : ceci inclut la fonctionnalité de la couche physique, mais aussi de la gestion des paquets échangés avec le FPGA, tel que cela a été prévu dans le protocole de communication. Le microcontrôleur et le FPGA

[10] Sanaa Seradni, stagiaire chez DreamWaferTM pendant la session d'été 2012

contiennent chacun un tableau de 16 registres de 8 bits (les registres contiennent des commandes ou des mesures). Toutes les 100 µs, l'ensemble des registres est mis à jour à travers le bus SPI cadencé à 10 MHz. Cette transmission à une durée de 20.8 µs (16 fois 1.3 µs). Pendant les 79.2 µs restantes (100-20.8), le microcontrôleur assure la communication sur le bus I2C.

- Communication I2C avec le système de refroidissement : la fonctionnalité de contrôle des ventilateurs avec le protocole IPMI a été validée. Le prototype actuel permet de contrôler la vitesse de rotation des ventilateurs à partir d'un des registres de commande partagés avec le FPGA.

Les prochaines étapes concernant le microcontrôleur PIC sont de valider le fonctionnement du code sur le Bottom PCB. De plus, il sera aussi nécessaire d'interfacer les cartes auxiliaires qui permettront d'avoir accès à de nouveaux capteurs nécessaires au fonctionnement du WaferBoard™ : capteurs de pression, capteurs magnétiques pour la fermeture du couvercle et systèmes de refroidissement additionnels. Le code C actuel correspondant à l'implémentation du microcontrôleur est d'environ 800 lignes.

B. Scripts Matlab

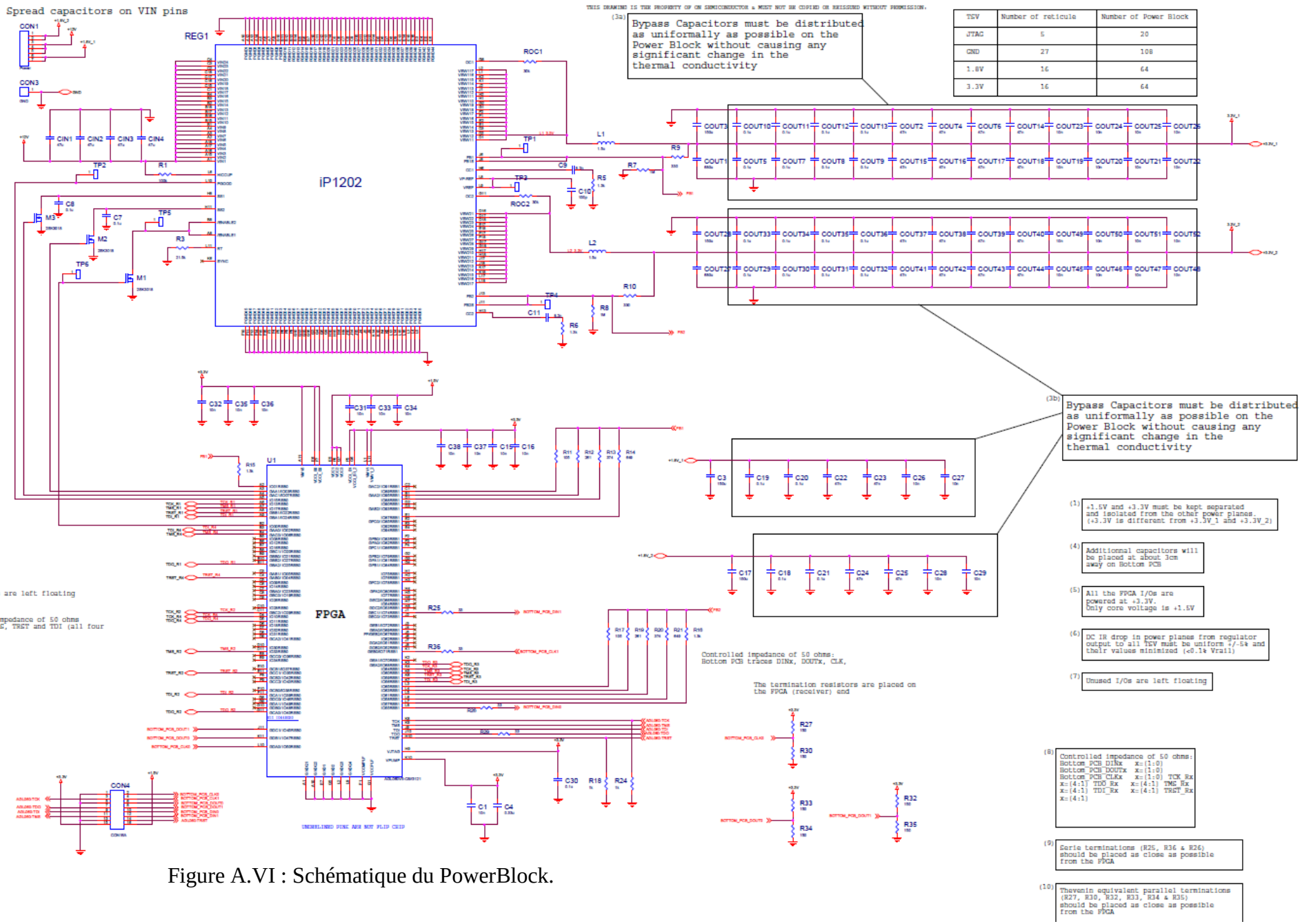
```
function [ res ] = freq_analysis_f( filename, N )

    %filename='freq1M.prn';
    M=importdata(filename);

    %nombre d'echantillons dans le fichier
    %N=2048;
    %taux de repetition (affichage)
    r=20;
    %periode d'échantillonnage (microsecondes)
    sample_period=1/156.25;

    i=0;
    j=0;

    %donnes d'analyse d'intégrité
    %tableau du nombre de zeros répétés
    repeated_z = ones([1,200]);
    count_z=0;
    Nz=0;
    %tableau du nombre de uns répétés
    repeated_o = ones([1,1000]);
    count_o=0;
    No=0;

    %EXTRACTION DES DONNEES
    x=M.data(1:N,1);
    y=M.data(1:N,3);
    valid=M.data(1:N,4);

    %FILTRAGE DES DONNEES INVALIDES
    i=1;
    while i<=N %pour chacune des données
        if(valid(i)==0) %si donnée non valide alors
            y(i)=[]; %on supprime l element des données
            valid(i)=[]; % et son bit valid associé
            N=N-1; %la taille du tableau diminue de 1 (passe automatiquement a la case suivante)
        else
            i=i+1; % sinon on passe a la case suivante
        end
    end

    %ANALYSE DE L'ECHANTILLONNAGE
    %on regarde le nombre de 1 et de 0 entre chaque front pour en déduire une
    %fréquence et un rapport cyclique
    for i = 1:N
        if(y(i)==0)
            count_z=count_z+1;
            if(count_o>0)
                No=No+1;
                repeated_o(No)=count_o;
```

Figure A.IV : Script d'analyse Matlab — Permet la reconstruction du signal échantillonné après la lecture en mémoire de l'outil de vérification.

```
            count_o=0;
        end
    else
        count_o=count_o+1;
        if(count_z>0)
            Nz=Nz+1;
            repeated_z(Nz)=count_z;
            count_z=0;
        end
    end
end
%le premier comptage n'est pas pris en compte car l'échantillonnage
%commence à n'importe quel moment..
repeated_o(1)=[];
repeated_z(1)=[];
No=No-1;
Nz=Nz-1;

%Analyse (microsecondes, megahertz)
low_period=mean(repeated_z(1:Nz))*sample_period;
high_period=mean(repeated_o(1:No))*sample_period;
period=low_period+high_period;
freq=1/period;
duty_cycle=high_period*100/period;

res = [low_period,high_period,period,freq,duty_cycle,No,Nz ];

%MISE EN FORME DES DONNEES POUR UN AFFICHAGE DE TYPE MODELSIM
x_extend=ones([1,N*r]);
y_extend=ones([1,N*r]);

for i = 1:N
    for j = 1:r
        x_extend((i-1)*r+j)=x(i)+j/r;
        y_extend((i-1)*r+j)=y(i);
    end
end
figure
hold on
plot(x_extend,y_extend)
%plot(x,y)
end

```

Figure A.V : Script d'analyse Matlab (suite) — Permet la reconstruction du signal échantillonné après la lecture en mémoire de l'outil de vérification.

C. Schématique du PowerBlock



TSV	Number of reticule	Number of Power Block
JTAG	5	20
GND	27	108
1.8V	16	64
3.3V	16	64

Figure A.VI : Schématique du PowerBlock.